ADSP-2136x SHARC® Processor Hardware Reference

Includes ADSP-21362, ADSP-21363, ADSP-21364, ADSP-21365, ADSP-21366

Revision 2.0, June 2009

Part Number
82-000501-01

Analog Devices, Inc.
One Technology Way
Norwood, Mass. 02062-9106

ANALOG DEVICES

Copyright Information

© 2008 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, Blackfin, EZ-KIT Lite, SHARC, the SHARC logo, TigerSHARC, and VisualDSP++ are registered trademarks of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

CONTENTS

PREFACE

Purpose of This Manual xxxi
Intended Audience xxxi
Manual Contents xxxii
What's New in This Manual xxxiv
Technical or Customer Support xxxvi
Supported Processors xxxvii
Product Information xxxvii
MyAnalog.com xxxviii
Processor Product Information xxxviii
Related Documents xxxix
Online Technical Documentation xl
Accessing Documentation From VisualDSP++ xl
Accessing Documentation From Windows xli
Accessing Documentation From the Web xli
Conventions xlii

INTRODUCTION

ADSP-2136x SHARC Design Advantages 1-1
SHARC Family Product Offerings 1-5
Processor Architectural Overview 1-6
Processor Core 1-7
Processor Peripherals 1-7
I/O Processor 1-7
Digital Applications Interface (DAI) 1-9
Development Tools 1-10
Architecture Enhancements 1-10
Parallel Port Enhancements 1-10
I/O Architecture Enhancements 1-11
Instruction Set Enhancements 1-11

I/O PROCESSOR

DMA Controller Operation 2-2
DMA Transfers Between Internal Memory 2-3
General Procedure for Configuring DMA 2-4
Summary 2-5
IOP Registers 2-5
Standard DMA Parameter Registers 2-6
Standard DMA Status Registers 2-8
Chaining DMA Status Registers 2-9
Data Buffers 2-9

DMA Channel Allocation 2-10
DMA Channel Priority 2-11
DMA Channel Arbitration Modes 2-14
Peripheral DMA Bus 2-14
Rotating Priority by Group 2-15
DMA Channel Interrupts 2-16
Internal Transfer Completion 2-16
DMA Channel Interrupt Priorities 2-16
Interrupt Versus Channel Priorities 2-17
DMA Controller Addressing 2-18
Internal Index Register Addressing 2-18
DMA Chaining 2-20
TCB Memory Storage 2-20
Chain Pointer Register 2-21
Chain Assignment 2-22
Starting Chain Loading 2-24
TCB Chain Loading Priority 2-25
Chain Insert Mode (SPORTs Only) 2-25
DMA Start and Stop Conditions 2-25
Configuring IOP/Core Interaction 2-26
Interrupt-Driven I/O 2-27
Polling DMA Channel Status 2-28
Standard DMA Status 2-29
Chaining DMA Status 2-29

TCB Storage 2-30
Serial Port TCB 2-30
Parallel Port TCB 2-30
SPI TCB 2-31
I/O Processor Register Access 2-32
IOP Access Conditions 2-32
Interrupt Latency 2-33
TCB Chain Loading Access 2-34
IOP Register Access Arbitration 2-34
IOP Performance 2-35

MEMORY-TO-MEMORY PORT DMA

Features 3-2
Functional Description 3-2
DMA Channels 3-2
Buffer 3-3
Interrupts 3-3
Data Throughput 3-3
Programming Model 3-4
Programming Example 3-4

PARALLEL PORT

Features 4-2
Pin Descriptions 4-3
Multiplexed Pin Functions 4-4

Functional Description 4-4
Multiplexed Operation 4-4
Address Cycles 4-4
Data Cycles 4-5
Data Buffers 4-6
Read Path 4-6
Write Path 4-7
Operation Modes 4-8
8-Bit Mode 4-8
16-Bit Mode 4-9
Parallel Port Registers 4-10
Control Register (PPCTL) 4-11
Data Buffer Register (RXPP/TXPP) 4-11
Data Transfer Types 4-11
DMA Transfers 4-12
DMA Internal Word Count Register (ICPP) 4-12
External Word Count Register (ECPP) 4-12
Chained DMA Transfers 4-13
DMA Chain Pointer Register (PPCP) 4-13
DMA Transfer Rules 4-13
Core-Driven Transfers 4-14
Interrupt Driven Accesses 4-15
Status-Driven Transfers (Polling) 4-16
Known-Duration Accesses 4-16

Core-Stall Driven Transfers 4-17
Interrupts 4-18
DMA Interrupts 4-19
Core Interrupts 4-19
Throughput 4-19
8-Bit Access 4-20
16-Bit Access 4-21
8-Bit Versus 16-Bit SRAM Modes 4-22
Parallel Port Effect Latency 4-23
Programming Model 4-23
Configuring the Parallel Port for DMA 4-24
Configuring a Chained DMA 4-24
Configuring the Parallel Port for Core Access 4-25
Programming Examples 4-26

DIGITAL APPLICATION INTERFACE

Features 5-1
Functional Description 5-3
Signal Naming Conventions 5-5
DAI Peripherals 5-6
I/O Pin Buffers 5-7
Pin Buffers as Signal Output 5-8
Pin Buffers as Signal Input 5-10
Programmable Pull-Up Resistors 5-11
Pin Buffers as Open Drain 5-11

Miscellaneous Buffers 5-12
Signal Routing Matrix by Groups 5-13
DAI Group Routing 5-14
Rules for SRU Connections 5-16
Making SRU Connections 5-16
Routing Capabilities 5-20
Default Routing 5-21
Interrupt Controller 5-24
System versus Exception Interrupts 5-24
Functional Description 5-25
Interrupt Channels 5-25
Interrupt Priorities 5-25
Miscellaneous Interrupts 5-26
Core versus DAI Interrupts 5-26
Interrupt Events 5-27
Servicing Interrupts 5-28
Debug Features 5-29
Shadow Registers 5-29
Loop Back Routing 5-30
Programming Model 5-31

SERIAL PORTS

Features 6-4
Pin Descriptions 6-6
SRU Configuration 6-6

SRU SPORT Receive Master 6-7
SRU SPORT Signal Integrity 6-7
Functional Description 6-9
Registers 6-10
Control Registers (SPCTLx) 6-12
Multichannel Control Registers (SPMCTLxy) 6-12
Data Buffers 6-13
Transmit Buffers (TXSPxA/B) 6-13
Transmit Path 6-14
Receive Buffers (RXSPxA/B) 6-14
Receive Path 6-15
Buffer Status 6-15
Multichannel Buffer Status 6-17
Selecting Operating Modes 6-18
Mode Selection 6-19
Data Word Formats 6-20
Word Length (SLEN) 6-20
Endian Format (LSBF) 6-21
Data Packing (PACK) 6-21
Data Type (DTYPE) 6-22
Companding the Data Stream 6-23
Companding As a Function 6-24
Clock Signal Options 6-25
Master Clock Divider Registers (DIVx) 6-25

Master Clock 6-25
Master Frame Sync 6-26
Slave Mode 6-26
Clock Source (ICLK, MSTR) 6-27
Sampling Edge (CKRE) 6-27
Frame Sync Options 6-28
Framed Versus Unframed Frame Syncs 6-28
Internal Versus External Frame Syncs (IFS, IMFS, MSTR) 6-29
Logic Level Frame Syncs (LFS, LMFS) 6-30
Early Versus Late Frame Syncs (LAFS) 6-31
Data-Independent Frame Sync (One Channel) 6-32
Data Independent Frame Sync (Two Channels) 6-33
Operating Modes 6-34
Standard Serial Mode 6-34
Timing Control Bits 6-34
Clocking Options 6-35
Frame Sync Options 6-35
Left-Justified Mode 6-36
Master Serial Clock and Frame Sync Rates 6-36
Left-Justified Mode Timing Control Bits 6-37
Frame Sync Channel First (L_FIRST) 6-37
I^2S Mode 6-38
I^2S Mode Timing Control Bits 6-39

Selecting Transmit and Receive Channel Order (L_FIRST) 6-39
Multichannel Operation 6-40
Frame Syncs Signals 6-42
Transmit Valid Signals 6-43
Multichannel Mode Control Bits 6-44
Number of Channels (NCH) 6-45
Frame Delay (MFD) 6-45
Channel Selection Registers 6-45
Companding Selection 6-46
Transmit Selection Registers 6-46
Receive Selection Registers 6-47
Data Transfer Types 6-47
Core Transfers 6-47
Single Word Transfers 6-47
Internal Memory DMA Transfers 6-48
Standard DMA 6-50
DMA Chaining 6-51
DMA Chain Insertion Mode 6-52
Interrupts 6-52
Internal Transfer Completion 6-52
Shared Channels 6-53
Debug Features 6-54
SPORT Loopback 6-54
Loopback Routing 6-55

Buffer Hang Disable (BHD) 6-55
Effect Latency 6-55
Programming Model 6-56
Setting Up and Starting Chained DMA 6-56
Enter DMA Chain Insertion Mode 6-57
Programming Examples 6-57

SERIAL PERIPHERAL INTERFACE PORTS

Features 7-1
Pin Descriptions 7-3
SRU Programming 7-4
Functional Description 7-5
Single Master Systems 7-7
Broadcast Access 7-8
Multi Master Systems 7-8
Register Descriptions 7-10
Control Register (SPICTLx) 7-10
Transfer Initiate Mode (TIMOD) 7-11
Input Slave Select Enable (ISSEN) 7-12
Disable MISO (DMISO) 7-12
Word Lengths (WL) 7-13
Modes (CLKPL, CPHASE) 7-14
Open Drain Mode (OPD) 7-17
Enable (SPIEN) 7-17
Packing (PACKEN) 7-18

Status Register (SPISTATx) 7-18
Multi Master Error (MME) 7-19
Transmission Error Bit (TUNF) 7-20
Reception Error Bit (ROVF) 7-20
Transmit Collision Error Bit (TXCOL) 7-20
BAUD Rate Register (SPIBAUDx) 7-21
DMA Control Register (SPIDMACx) 7-21
Data Transfer Types 7-21
Core Transfers 7-22
DMA Transfers 7-22
Slave DMA Transfer Preparation 7-24
DMA Chaining 7-25
Setting Up and Starting Chained DMA 7-25
Core and DMA Transfers 7-26
Changing Configuration 7-26
Starting and Stopping Data Transfers 7-27
Interrupts 7-28
Interrupt Sources 7-28
Internal Transfer Completion 7-30
DMA Error Interrupts 7-30
Slave Select Timing 7-31
Debug Features 7-32
Shadow Register 7-32
Internal Loopback Mode 7-33

Loopback Routing 7-33
Programming Model 7-33
Master Mode Core Transfers 7-33
Slave Mode Core Transfers 7-35
Master Mode DMA Transfers 7-36
Slave Mode DMA Transfers 7-38
Chained DMA Transfers 7-39
Stopping Core Transfers 7-40
Stopping DMA Transfers 7-40
Switching from Transmit To Transmit/Receive DMA 7-41
Switching from Receive to Receive/Transmit DMA 7-42
Programming Examples 7-44

INPUT DATA PORT

Features 8-1
Functional Description 8-3
Serial Input Port 8-4
Pin Descriptions 8-5
SRU Descriptions 8-5
Input Data Formats 8-6
Register Descriptions 8-9
Control Registers (IDP_CTLx) 8-9
Status Registers (DAI_STAT0) 8-9
Parallel Data Acquisition Port (PDAP) 8-9
Port Selection 8-10

Pin Descriptions 8-11
SRU Programming 8-12
Register Descriptions 8-12
Control Register (IDP_PP_CTL) 8-12
PDAP Data Packing 8-13
Mode 11 (No Packing) 8-13
Mode 10 (Packing by 2) 8-14
Mode 01 (Packing by 3) 8-14
Mode 00 (Packing by 4) 8-14
Timing 8-15
Data Buffer 8-17
Data Transfer Types 8-18
Core Transfers 8-18
DMA Transfers 8-19
DMA Channel Priority 8-19
Standard DMA 8-20
Ping-Pong DMA 8-20
Data Input Format 8-21
Multichannel DMA Operation 8-22
Multichannel FIFO Status 8-22
Interrupts 8-23
Core FIFO Threshold Interrupts 8-23
DMA Interrupts 8-23
FIFO Overflow Interrupts 8-24

Servicing Interrupts 8-24
Debug Features 8-24
Buffer Hang Disable 8-25
Shadow Registers 8-25
Core FIFO Write 8-25
IDP Effect Latency 8-26
Programming Model 8-26
Setting Miscellaneous Bits 8-26
Starting Core Interrupt-Driven Transfer 8-27
Starting A Standard DMA Transfer 8-27
Starting a Ping-Pong DMA Transfer 8-28
Servicing Interrupts for DMA 8-29
Programming Example 8-31

PERIPHERAL TIMERS

Features 9-1
Pin Descriptions 9-3
SRU Programming 9-4
Functional Description 9-4
Register Descriptions 9-5
Count Registers 9-5
Counter Registers (TMxCNT) 9-6
Period Registers (TMxPRD) 9-6
Pulse Width Register (TMxW) 9-6
Status and Control Registers 9-7

Operation 9-7
Mode Selection 9-8
Pulse Width Modulation Mode (PWM_OUT) 9-9
PWM Waveform Generation 9-12
Single-Pulse Generation 9-13
Pulse Mode 9-13
Pulse Width Count and Capture Mode (WDTH_CAP) 9-14
External Event Watchdog Mode (EXT_CLK) 9-17
Interrupts 9-19
Sources 9-19
Watchdog Functionality 9-20
Effect Latency 9-20
Debug Features 9-21
Loopback Routing 9-21
Programming Model 9-22
PWM Out Mode 9-22
WDTH_CAP Mode 9-23
EXT_CLK Mode 9-24
Programming Examples 9-25

PULSE WIDTH MODULATION

Features 10-1
Pin Descriptions 10-4
Functional Description 10-4
Register Descriptions 10-5

Operation Modes 10-6
Groups Synchronization 10-6
PWM Timer 10-7
Edge-Aligned Mode 10-8
Center-Aligned Mode 10-9
Switching Frequencies 10-11
Dead Time 10-12
Duty Cycles 10-13
Duty Cycles and Dead Time 10-13
Over-Modulation 10-18
Update Modes 10-20
Single-Update 10-20
Double-Update 10-20
Configuring Polarity 10-20
Accuracy 10-21
Duty Cycle 10-22
Output Enable 10-22
Crossover Mode 10-23
Interrupts 10-24
Debug Features 10-24
Programming Example 10-24

SONY/PHILIPS DIGITAL INTERFACE

Features 11-2
S/PDIF Transmitter 11-3

Pin Descriptions 11-3
SRU Programming 11-4
Functional Description 11-5
Input Data Format 11-7
Output Data Mode 11-8
Operation Modes 11-9
Standalone Mode 11-9
Full Serial Mode 11-9
Register Descriptions 11-9
Control Register (DITCTL) 11-9
Channel Status Registers (DITCHANAx/Bx) 11-10
S/PDIF Receiver 11-10
Pin Descriptions 11-11
SRU Programming 11-12
Functional Description 11-13
Output Data Format 11-13
PLL Selection for Clock Recovery 11-13
Register Descriptions 11-14
Receiver Control Register (DIRCTL) 11-14
Receiver Status Register (DIRSTAT) 11-15
Clock Recovery 11-17
Channel Decoding 11-17
Channel Status 11-17
Compressed or Non-linear Audio Data 11-18

Emphasized Audio Data 11-19
Single-Channel Double-Frequency Mode 11-19
Interrupts 11-20
Transmitter Interrupt 11-20
Receiver Interrupts 11-20
Debug Features 11-21
Loopback Routing 11-21
Programming Model 11-21
Programming the Transmitter 11-21
Programming the Receiver 11-22
Interrupted Data Streams on the Receiver 11-23

ASYNCHRONOUS SAMPLE RATE CONVERTER

Features 12-1
Pin Descriptions 12-3
SRU Programming 12-3
Functional Description 12-4
Serial Data Ports 12-8
De-Emphasis Filter 12-9
Mute Control 12-9
Automatic Mute 12-10
Soft Mute 12-10
Hard Mute 12-10
Auto Mute 12-10
Register Descriptions 12-11

Control Registers (SRCCTLn) 12-11
Data Format 12-12
Word Width 12-12
Ratio Registesr (SRCRATx) 12-13
Operation Modes 12-13
TDM Daisy Chain Mode 12-14
TDM Output Daisy Chain 12-14
TDM Input Daisy Chain 12-15
Bypass Mode 12-16
Matched-Phase Mode (ADSP-21364 Only) 12-16
Data Format Matched-Phase Mode 12-18
Group Delay 12-18
Interrupts 12-19

PRECISION CLOCK GENERATOR

Features 13-1
Functional Description 13-2
Pin Descriptions 13-4
SRU Programming 13-5
Register Descriptions 13-5
Clock Inputs 13-6
Clock Outputs 13-6
Frame Sync Outputs 13-7
Normal Mode 13-7
Bypass Mode 13-8

External Trigger Mode 13-9
Frame Sync 13-10
Phase Shift 13-11
Phase Shift Settings 13-11
Pulse Width 13-13
Bypass Mode 13-13
Bypass as a Pass Through 13-14
Bypass as a One-Shot 13-14
Programming Examples 13-16
Setup for I^2S or Left-Justified DAI 13-16
Channel B Clock and Frame Sync Divisors 13-21
Channel A and B Output Example 13-23

SYSTEM DESIGN

Conditioning Input Signals 14-2
Reset Input Hysteresis 14-2
Clock Input Specifications and Jitter 14-3
Input Synchronization Delay 14-3
Clocking 14-4
Input Clock 14-5
Input Clock Divider 14-6
Feedback Divider 14-6
Hardware Control 14-7
Software Control 14-8
VCO Clock 14-8

Output Clock Generator 14-9
Core Clock (CCLK) 14-9
Peripheral Clock (PCLK) 14-10
Bypass Clock 14-10
Power Savings 14-10
Power Supplies 14-11
Power Supply for the PLL 14-11
Power-Up Sequence 14-11
Input Clock 14-11
PLL Start-Up 14-12
Examples for Power Management 14-13
Example for Output Divider Management 14-13
Examples For VCO Clock Management 14-14
RESET Function 14-16
Processor Pin Descriptions 14-16
JTAG Interface Pins 14-16
Pin Impedance 14-17
Pin Multiplexing 14-17
FLAG3–0 Pins 14-18
Parallel Port Pin Multiplexing 14-20
SPI Master Slave Select 14-23
Parallel Port and DAI Pin Multiplexing Scheme 14-24
PWM Multiplexing Scheme 14-24
SRU Flag Description 14-25

System Components 14-26
Supervisory Circuits 14-26
Designing for High Frequency Operation 14-28
Other Recommendations and Suggestions 14-29
Decoupling and Grounding 14-29
Oscilloscope Probes 14-30
Recommended Reading 14-31
Processor Booting 14-32
Boot Mechanisms 14-32
Booting Process 14-32
Loading the Boot Kernel Using DMA 14-33
Executing the Boot Kernel 14-33
Loading the Application 14-33
Loading the Application's Interrupt Vector Table 14-34
Starting Program Execution 14-34
Internal Memory Kernel Load 14-35
Parallel Port Booting 14-36
SPI Port Booting 14-38
Master Boot Mode 14-38
Master Header Information 14-40
Slave Boot Mode 14-41
SPI Boot Packing 14-42
32-Bit SPI Packing 14-44
16-Bit SPI Packing 14-45

8-Bit SPI Packing 14-46
Kernel Boot Time 14-47
Definition of Terms 14-48

REGISTERS REFERENCE

I/O Processor Registers A-2
Notes on Reading Register Drawings A-3
System Control Register (SYSCTL) A-4
Power Management Control Register (PMCTL) A-6
Peripheral Registers A-10
MTM DMA Control (MTMCTL Register) A-10
Parallel Port Registers A-10
Parallel Port DMA Registers A-10
Parallel Port Control Register (PPCTL) A-11
Serial Peripheral Interface Registers A-14
SPI Control Registers (SPICTL, SPICTLB) A-14
DMA Configuration Registers (SPIDMAC, SPIDMACB) . A-18
SPI Baud Rate Registers (SPIBAUD, SPIBAUDB) A-20
SPI Port Status (SPISTAT, SPISTATB) Registers A-20
SPI Port Flags Registers (SPIFLG, SPIFLGB) A-22
RXSPI Shadow Registers (RXSPI_SHADOW, RXSPIB_SHADOW) A-23
Pulse Width Modulation Registers A-23
PWM Global Control Register (PWMGCTL) A-23

PWM Global Status Register (PWMGSTAT) A-24
PWM Control Register (PWMCTLx) A-25
PWM Status Registers (PWMSTATx) A-26
PWM Period Registers (PWMPERIODx) A-27
PWM Output Disable Registers (PWMSEGx) A-27
PWM Polarity Select Registers (PWMPOLx) A-28
PWM Channel Duty Control Registers (PWMAx, PWMBx) A-29
PWM Channel Low Duty Control Registers (PWMALx, PWMBLx) A-29
PWM Dead Time Registers (PWMDTx) A-29
PWM Debug Status Registers (PWMDBGx) A-30
Peripherals Routed Through the DAI A-30
Serial Port Registers A-30
SPORT Serial Control Registers (SPCTLx) A-30
SPORT Multichannel Control Registers (SPMCTLxy) A-43
SPORT Transmit Select Registers (MTxCSy) A-45
SPORT Transmit Compand Registers (MTxCCSy) A-45
SPORT Receive Select Registers (MRxCSx) A-46
SPORT Receive Compand Registers (MRxCCSx) A-46
SPORT Divisor Registers (DIVx) A-46
Input Data Port Registers A-47
Input Data Port DMA Control Registers A-47
Input Data Port Control Register 0 (IDP_CTL0) A-47
Input Data Port Control Register 1 (IDP_CTL1) A-49

Parallel Data Acquisition Port Control Register (IDP_PP_CTL) A-50
Input Data Port FIFO Register (IDP_FIFO) A-53
IDP Status Register (DAI_STAT0) A-54
IDP Status Register 1 (DAI_STAT1) A-56
Peripheral Timer Registers A-56
Timer Configuration Registers (TMxCTL) A-56
Timer Status Registers (TMxSTAT) A-57
Sample Rate Converter Registers A-59
SRC Control Registers (SRCCTLx) A-59
SRC Mute Register (SRCMUTE) A-64
SRC Ratio Registers (SRCRATx) A-64
Precision Clock Generator Registers A-65
Control Registers (PCG_CTLxy) A-65
Pulse Width Register (PCG_PW) A-67
Synchronization Register (PCG_SYNC) A-69
Sony/Philips Digital Interface Registers A-70
Transmitter Registers A-70
Transmit Control Register (DITCTL) A-70
Transmit Status Registers for Subframe A (DITCHANL) A-72
Transmit Status Registers for Subframe B (DITCHANR) A-73
Receiver Registers A-73
Receive Control Register (DIRCTL) A-73

Receive Status Register (DIRSTAT) A-75
Receive Status Registers for Subframe A (DIRCHANL) A-77
Receive Status Registers for Subframe B (DIRCHANR) A-77
DAI Interrupt Controller Registers A-77
DAI Status Register A-78
Digital Applications Interface Status Register (DAI_STAT) A-79
DAI Signal Routing Unit Registers A-80
Clock Routing Control Registers (SRU_CLKx, Group A) A-80
Serial Data Routing Registers (SRU_DATx, Group B) A-85
Frame Sync Routing Control Registers (SRU_FSx, Group C) A-89
Pin Signal Assignment Registers (SRU_PINx, Group D) A-93
Miscellaneous Signal Routing Registers (SRU_MISCx, Group E) A-98
DAI Pin Buffer Enable Registers (SRU_PBENx, Group F) A-102
DAI Status Registers A-106
DAI Pin Buffer Registers (DAI_PIN_PULLUP, DAI_PIN_STAT) A-106
Register Listing A-108

INTERRUPTS

Programmable Interrupt Control Registers B-1
Programmable Interrupt Control Register 0 (PICR0) B-3
Programmable Interrupt Control Register 1 (PICR1) B-4
Programmable Interrupt Control Register 2 (PICR2) B-5
Programmable Interrupt Control Register 3 (PICR3) B-5

AUDIO FRAME FORMATS

Overview C-2
Standard Serial Mode C-2
I^2S Mode C-3
Left-Justified Mode C-5
Right-Justified Mode C-5
TDM Mode C-6
Packed TDM Mode C-7
MOST Mode C-8
AES/EBU/SPDIF Formats C-9
Subframe Format C-11
Channel Coding C-14
Preambles C-15

INDEX

PREFACE

Thank you for purchasing and developing systems using ADSP-21362, ADSP-21363, ADSP-21364, ADSP-21365, ADSP-21366, SHARC® processors from Analog Devices.

Purpose of This Manual

The *ADSP-2136x SHARC Processor Hardware Reference for the ADSP-21362/3/4/5/6 Processors* contains information about the peripheral set and I/O properties for these products. These are 32-bit, fixed- and floating-point digital signal processors from Analog Devices for use in computing, communications, and consumer applications.

The manual provides information on the processor's I/O architecture and the operation of the peripherals associated with each model.

Intended Audience

The primary audience for this manual is a programmer who is familiar with Analog Devices processors. This manual assumes that the audience has a working knowledge of the appropriate processor architecture and instruction set. Programmers who are unfamiliar with Analog Devices processors can use this manual, but should supplement it with other texts (such as the appropriate hardware reference manuals and data sheets) that describe your target architecture.

Manual Contents

This manual provides detailed information about the ADSP-2136x processors in the following chapters:

- Chapter 1, "Introduction"
  Provides an architectural overview of the ADSP-2136x SHARC processors.
- Chapter 2, "I/O Processor"
  Describes input/output processor architecture, and provides direct memory access (DMA) procedures for the processor peripherals.
- Chapter 2, "Memory-to-Memory Port DMA"
  The memory-to-memory DMA controller is capable of transferring 64-bit bursts of data between internal memories.
- Chapter 4, "Parallel Port"
  Describes how the processor's on-chip DMA controller acts as a machine for transferring data without core interruption.
- Chapter 5, "Digital Application Interface"
  Provides information about the digital application interface (DAI) which allows you to attach an arbitrary number and variety of peripherals to the processor while retaining high levels of compatibility.
- Chapter 6, "Serial Ports"
  Describes the six dual data line serial ports. Each SPORT contains a clock, a frame sync, and two data lines that can be configured as either a receiver or transmitter pair.
- Chapter 7, "Serial Peripheral Interface Ports"
  Describes the operation of the serial peripheral interface (SPI) port. SPI devices communicate using a master-slave relationship and can achieve high data transfer rate because they can operate in full-duplex mode.

- Chapter 8, "Input Data Port"
  Discusses the function of the input data port (IDP) which provides a low overhead method of routing signal routing unit (SRU) signals back to the core's memory.
- Chapter 9, "Peripheral Timers"
  The processor processors contain three identical 32-bit timers that can be used to interface with external devices.
- Chapter 10, "Pulse Width Modulation"
  Describes the implementation and use of the pulse width modulation module which provides a technique for controlling analog circuits with the microprocessor's digital outputs.
- Chapter 11, "Sony/Philips Digital Interface"
  Provides information on the use of the Sony/Philips Digital Interface which is a standard audio file transfer format that allows the transfer of digital audio signals from one device to another without having to be converted to an analog signal.
- Chapter 12, "Asynchronous Sample Rate Converter"
  Provides information on the sample rate converter (SRC) module. This module performs synchronous or asynchronous sample rate conversion across independent stereo channels, without using any internal processor resources.
- Chapter 13, "Precision Clock Generator"
  Details the precision clock generators (PCG), each of which generates a pair of signals derived from a clock input signal.
- Chapter 14, "System Design"
  Describes system design features of the ADSP-2136x processor. These include power, reset, clock, JTAG, and booting, as well as pin multiplexing schemes and other system-level information.

- Appendix A, "Registers Reference"
  Provides a graphical presentation of all registers and describes the bit usage in each register.
- Appendix B "Interrupts"
  Provides information on the programmable interrupt control registers (PICRx). These registers allow programs to substitute the default interrupts for some other interrupt source.
- Appendix C "Audio Frame Formats"
  Provides an overview of the formats available to designers of audio applications through the various peripherals of the SHARC processors.

This hardware reference is a companion document to the *SHARC Processor Programming Reference*.

What's New in This Manual

This is the second edition (Revision 2.0) of the *ADSP-2136x SHARC Processor Hardware Reference*. The following changes should be noted.

- All document errata for this manual has been corrected.
- Changed digital audio interface to digital applications interface throughout this document. This change is a naming convention change only and does not effect the operation or specification of this product in any way.
- Added "Peripheral Timers" chapter. This chapter was incorrectly placed in the *SHARC Processor Programming Reference*.
- All chapters of this document have been revised to correct errors (not reported through the errata system), to remove redundant information, and to reorganize information so that it is presented logically and consistently.

Where appropriate, the chapters contain the following information, presented in this order:

- Primary features
- Hardware interface (pins)
- Basic function of the peripheral
- Primary registers used by this peripheral
- Basic peripheral operation, including DMA
- Debug features
- Programming model (illustrates programming sequences)
- Programming example(s)

- A new table has been created and placed at the beginning of each chapter that provides a list of theprimary features contained in that peripheral. This is intended to provide an “at-a-glance” summary so that you may quickly decide whether this module is desirable for your particular design.
- A new mode has been added to the PWM module. See “Crossover Mode” on page 10-23.
- “Register Listing” on page A-108 provides a listing of all registers that are user accessible for the SHARC processor and includes the register mnemonic, address, description, and reset state.
- A new appendix, “Audio Frame Formats” on page C-1, provides an overview of the audio standards that are available on the SHARC processors. This appendix consolidates information that was spread throughout the previous manual.

- The core related interrupt register descriptions have been moved to the *SHARC Processor Programming Reference*. These registers are the Interrupt Latch register (IRPTL), Interrupt Mask register (IMASK), Interrupt Mask Pointer register (IMASKP), Interrupt register (LIRPTL), Mode Mask register (MMASK).

Technical or Customer Support

You can reach Analog Devices, Inc. Customer Support in the following ways:

- Visit the Embedded Processing and DSP products Web site at
  http://www.analog.com/processors/technicalSupport
- E-mail tools questions to
  processor.tools.support@analog.com
- E-mail processor questions to
  processor.support@analog.com (World wide support)
  processor.europe@analog.com (Europe support)
  processor.china@analog.com (China support)
- Phone questions to **1-800-ANALOGD**
- Contact your Analog Devices, Inc. local sales office or authorized distributor
- Send questions by mail to:

  Analog Devices, Inc.
  One Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106
  USA

Supported Processors

The following is the list of Analog Devices, Inc. processors supported in VisualDSP++®.

TigerSHARC® (ADSP-TSxxx) Processors

The name *TigerSHARC* refers to a family of floating-point and fixed-point [8-bit, 16-bit, and 32-bit] processors. VisualDSP++ currently supports the following TigerSHARC families: ADSP-TS101 and ADSP-TS20x.

SHARC (ADSP-21xxx) Processors

The name *SHARC* refers to a family of high-performance, 32-bit, floating-point processors that can be used in speech, sound, graphics, and imaging applications. VisualDSP++ currently supports the following SHARC families: ADSP-2106x, ADSP-2116x, ADSP-2126x, ADSP-2136x, and ADSP-2137x.

Blackfin® (ADSP-BFxxx) Processors

The name *Blackfin* refers to a family of 16-bit, embedded processors. VisualDSP++ currently supports the following Blackfin families: ADSP-BF53x and ADSP-BF56x.

Product Information

You can obtain product information from the Analog Devices Web site. Analog Devices is online at `www.analog.com`. Our Web site provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

MyAnalog.com

MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information on products you are interested in. You can also choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Registration

Visit www.myanalog.com to sign up. Click **Register** to use MyAnalog.com. Registration takes about five minutes and serves as a means to select the information you want to receive.

If you are already a registered user, just log on. Your user name is your e-mail address.

Processor Product Information

For information on embedded processors and DSPs, visit our Web site at www.analog.com/processors, which provides access to technical publications, data sheets, application notes, product overviews, and product announcements.

You may also obtain additional information about Analog Devices and its products in any of the following ways.

- E-mail questions or requests for information to
  processor.support@analog.com (World wide support)
  processor.europe@analog.com (Europe support)
  processor.china@analog.com (China support)

- Fax questions or requests for information to
  1-781-461-3010 (North America)
  +49-89-76903-157 (Europe)
- Access the FTP Web site at
  `ftp ftp.analog.com` (or `ftp 137.71.25.69`)
  `ftp://ftp.analog.com`

Related Documents

The following publications that describe the ADSP-2136x processor (and related processors) are available online:

- *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*
- *ADSP-21367/8/9 SHARC Processor Data Sheet*
- *ADSP-21371/ADSP-21375 SHARC Processor Data Sheet*
- *SHARC Processor Programming Reference*

For information on product related development software and Analog Devices processors, see these publications:

- *VisualDSP++ User's Guide*
- *VisualDSP++ C/C++ Compiler and Library Manual*
- *VisualDSP++ Assembler and Preprocessor Manual*
- *VisualDSP++ Linker and Utilities Manual*
- *VisualDSP++ Kernel (VDK) User's Guide*

Visit the Technical Library Web site to access all processor and tools manuals and data sheets:

`http://www.analog.com/processors/technical_library`

Online Technical Documentation

Online documentation comprises the VisualDSP++ Help system, software tools manuals, hardware tools manuals, processor manuals, the Dinkum Abridged C++ library, and Flexible License Manager (FlexLM) network license manager software documentation. You can easily search across the entire VisualDSP++ documentation set for any topic of interest. For easy printing, supplementary `.PDF` files of most manuals are also provided.

Each documentation file type is described as follows.

File D	escription
`.CHM`	Help system files and manuals in Help format
`.HTM` or `.HTML`	Dinkum Abridged C++ library and FlexLM network license manager software documentation. Viewing and printing the `.HTML` files requires a browser, such as Internet Explorer 4.0 (or higher).
`.PDF`	VisualDSP++ and processor manuals in Portable Documentation Format (PDF). Viewing and printing the `.PDF` files requires a PDF reader, such as Adobe Acrobat Reader (4.0 or higher).

If documentation is not installed on your system as part of the software installation, you can add it from the VisualDSP++ CD-ROM at any time by running the tools installation. Access the online documentation from the VisualDSP++ environment, Windows® Explorer, or the Analog Devices web site.

Accessing Documentation From VisualDSP++

From the VisualDSP++ environment:

- Access VisualDSP++ online Help from the Help menu's **Contents**, **Search**, and **Index** commands.
- Open online Help from context-sensitive user interface items (toolbar buttons, menu commands, and windows).

Accessing Documentation From Windows

In addition to any shortcuts you may have constructed, there are many ways to open VisualDSP++ online Help or the supplementary documentation from Windows.

Help system files (.CHM) are located in the Help folder, and PDF files are located in the Docs folder of your VisualDSP++ installation CD-ROM. The Docs folder also contains the Dinkum Abridged C++ library and the FlexLM network license manager software documentation.

Using Windows Explorer

- Double-click the vdsp-help.chm file, which is the master Help system, to access all the other .CHM files.
- Double-click any file that is part of the VisualDSP++ documentation set.

Using the Windows Start Button

- Access VisualDSP++ online Help by clicking the **Start** button and choosing **Programs**, **Analog Devices**, **VisualDSP++**, and **VisualDSP++ Documentation**.
- Access the .PDF files by clicking the **Start** button and choosing **Programs**, **Analog Devices**, **VisualDSP++**, **Documentation for Printing**, and the name of the book.

Accessing Documentation From the Web

Download manuals at the following Web site:
http://www.analog.com/processors/technical_library

Select a processor family andbook title. Download archive (ZIP) files, one for each manual. Use any archive management software, such as WinZip, to decompress downloaded files.

Conventions

Text conventions used in this manual are identified and described as follows.

Example	Description
Close command (**File** menu)	Titles in reference sections indicate the location of an item within the VisualDSP++ environment's menu system (for example, the **Close** command appears on the **File** menu).
`{this | that }`	Alternative items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. One or the other is required.
`[this | that ]`	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional `this` or `that`.
`[this,…]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `letter gothic` font.
`filename`	Non-keyword placeholders appear in text with italic style format.
	Note: For correct operation, ... A Note: provides supplementary information on a related topic. In the online version of this book, the word **Note** appears instead of this symbol.
	Caution: Incorrect device operation may result if ... **Caution:** Device damage may result if ... A Caution: identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word **Caution** appears instead of this symbol.
	Warning: Injury to device users may result if ... A Warning: identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word **Warning** appears instead of this symbol.

Additional conventions, which apply only to specific chapters, may appear throughout this document.

1 INTRODUCTION

The ADSP-2136x SHARC processors are high performance 32-bit processors used for high quality audio, medical imaging, communications, military, test equipment, 3D graphics, speech recognition, motor control, imaging, and other applications. By adding on-chip SRAM, integrated I/O peripherals, and an additional processing element for single-instruction, multiple-data (SIMD) support, this processor builds on the ADSP-21000 family DSP core to form a complete system-on-a-chip.

ADSP-2136x SHARC Design Advantages

A digital signal processor's data format determines its ability to handle signals of differing precision, dynamic range, and signal-to-noise ratios. Because floating-point DSP math reduces the need for scaling and the probability of overflow, using a floating-point processor can simplify algorithm and software development. The extent to which this is true depends on the floating-point processor's architecture. Consistency with IEEE workstation simulations and the elimination of scaling are clearly two ease-of-use advantages. High level language programmability, large address spaces, and wide dynamic range allow system development time to be spent on algorithms and signal processing concerns, rather than assembly language coding, code paging, and error handling. The ADSP-2136x processor is a highly integrated, 32-bit floating-point processor which provides all of these design advantages.

The SHARC processor architecture balances a high performance processor core with high performance program memory (PM), data memory (DM) and input/output (I/O) buses. In the core, every instruction can execute in

a single cycle. The buses and instruction cache provide rapid, unimpeded data flow to the core, thereby maintaining the execution rate.

The ADSP-2136x processors contain the following architectural features:

- Two processing elements (PEx and PEy), each containing 32-bit IEEE floating-point computation units—multiplier, arithmetic logic unit (ALU), shifter, and data register file.
- Program sequencer with related instruction cache, timer, core timer, and data address generators (DAG1 and DAG2).
- 3M bits of SRAM.
- Parallel port for interfacing to off-chip memory and peripherals.
- IOP with integrated direct memory access (DMA) controllers for each DMA channel, serial peripheral interface (SPI) compatible port, and serial ports (SPORTs) for point-to-point multiprocessor communications.
- A variety of audio-centric peripheral modules including a Sony/Philips Digital Interface (S/PDIF), sample rate converter (SRC) and pulse width modulation (PWM). The Digital Transmission Content Protection protocol (DTCP) is available on the ADSP-21365 and ADSP-21362 processors. Table 1-1 on page 1-6 provides details on these as well as other features for the current members of the ADSP-2136x processor family.
- JTAG test access port (TAP) for emulation.

The ADSP-2136x processor also contains three on-chip buses: PM bus, DM bus, and I/O bus. The PM bus provides access to either instructions or data. During a single cycle, these buses let the processor access two data operands from memory, access an instruction (from the cache), and perform a DMA transfer.

Figure 1-1 illustrates a typical single processor system.

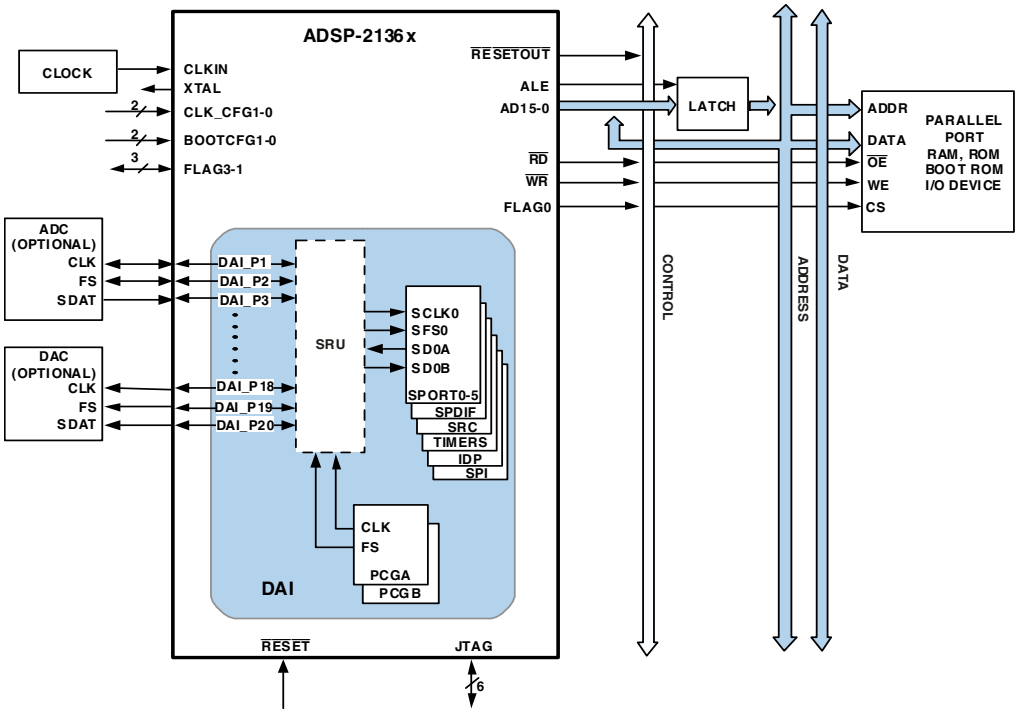


Figure 1-1. ADSP-2136x Processor Typical Single Processor System

The ADSP-2136x processors address the five central requirements for signal processing:

1. **Fast, Flexible Arithmetic.** The ADSP-21000 family processors execute all instructions in a single cycle. They provide fast cycle times and a complete set of arithmetic operations. The processor is IEEE floating-point compatible and allows either interrupt on arithmetic exception or latched status exception handling.

2. **Unconstrained Data Flow.** The processor has a Super Harvard Architecture combined with a ten-port data register file. In every cycle, the processor can write or read two operands to or from the

register file, supply two operands to the ALU, supply two operands to the multiplier, and receive three results from the ALU and multiplier. The processor's 48-bit orthogonal instruction word supports parallel data transfers and arithmetic operations in the same instruction.

3. **Computation Precision.** The processor handles 32-bit IEEE single precision or 40-bit IEEE extended precision floating-point formats. The processors can carry 32-bit integer and fractional formats (twos-complement and unsigned), throughout their computation units, limiting intermediate data truncation errors (up to 80 bits of precision are maintained during multiply-accumulate operations).

4. **Dual Address Generators.** The processor has two data address generators (DAGs) that provide immediate or indirect (pre- and post-modify) addressing. Modulus, bit-reverse, and broadcast operations are supported with no constraints on data buffer placement.

5. **Efficient Program Sequencing.** In addition to zero-overhead loops, the processor supports single-cycle setup and exit for loops. Loops are both nestable (six levels in hardware) and interruptable. The processors support both delayed and non-delayed branches.

The ADSP-2136x processors also provide the following features, increasing the variety of applications that these processors can be used for.

- **High bandwidth I/O.** The processors contain a dedicated, 4M bits on-chip ROM, a parallel port, an SPI port, serial ports, digital application interface (DAI), and JTAG.

- **Serial ports.** Provides an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices.

- **digital application interface (DAI).** The DAI includes a precision clock generator, an input data port, SPORT, and SPI.

- **Input data port (IDP).** The IDP provides an additional input path to the processor core, configurable as eight channels of serial data or seven channels of serial data and a single channel of up to 20-bit wide parallel data.
- **Signal routing unit (SRU).** The SRU is part of the DAI and provides configuration flexibility by allowing software-programmable connections to be made between the DAI components.
- **Two serial peripheral interfaces (SPI).** The primary SPI has dedicated pins and the secondary is controlled through the DAI. The SPI provides master or slave serial boot through the SPI, full-duplex operation, master-slave mode, multi-master support, open drain outputs, programmable baud rates, clock polarities, and phases.
- **I/O processor (IOP).** The IOP manages the SHARC processor's off-chip data I/O to alleviate the core of this burden. This unit manages the other processor peripherals as well as direct memory accesses (DMA).

SHARC Family Product Offerings

Table 1-1 provides information on the products covered in this manual. Note that each model is available for automotive applications with controlled manufacturing. Note that these special models may have specifications that differ from the general release models. For information on which models are available as automotive, see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

Table 1-1. SHARC Processor Family Features

Feature	ADSP-21362[1]	ADSP-21363	ADSP-21364	ADSP-21365[1]	ADSP-21366
RAM	3M bit				
ROM	4M bit				
Audio Decoders in ROM[2]	No	No	No	Yes	Yes
S/PDIF	Yes	No	Yes	Yes	Yes
SRC Performance	128db	No SRC	140dB	128dB	128dB
Package Option[3]	136-ball BGA 144-lead LQFP, exposed pad				
Processor Speed	333 MHz				

1 The ADSP-21362 and ADSP-21365 processors provide the Digital Transmission Content Protection protocol, a proprietary security protocol. Contact your Analog Devices sales office for more information.

2 Audio decoding algorithms include PCM, Dolby Digital EX, Dolby Prologic IIx, DTS 96/24, Neo:6, DTS ES, MPEG2 AAC, MP3, and functions like Bass management, Delay, Speaker equalization, Graphic equalization, and more. Decoder/post-processor algorithm combination support varies depending upon the chip version and the system configurations. Please visit www.analog.com/SHARC for complete information.

3 Analog Devices offers these packages in RoHS compliant versions.

Processor Architectural Overview

The ADSP-2136x processor forms a complete system-on-a-chip, integrating a large, high speed SRAM and I/O peripherals supported by a dedicated I/O bus. The following sections summarize the features of each functional block.

Processor Core

The processor core of the processor consists of two processing elements (each with three computation units and data register file), a program sequencer, two data address generators, a timer, and an instruction cache. All digital signal processing occurs in the processor core. For complete information, see the *SHARC Processor Programming Reference*.

Processor Peripherals

The term processor peripherals refers to the multiple on-chip functional blocks used to communicate with off-chip devices. The ADSP-2136x peripherals include the JTAG, parallel, serial, SPI ports, DAI components (PCG, timers, and IDP), and any external devices that connect to the processor.

I/O Processor

The input/output processor (IOP) manages the processor's off-chip data I/O to alleviate the core of this burden. Up to 25 simultaneous DMA transfers (25 DMA channels) are supported for transfers between ADSP-2136x processor internal memory and serial ports (12), the input data port (IDP) (8), SPI port (2), and the parallel port. The I/O processor can perform DMA transfers between the peripherals and internal memory at the core/2 clock speed. The architecture of the internal memory allows the IOP and the core to access internal memory simultaneously (assuming no block conflicts) with no reduction in throughput.

Serial ports. The ADSP-2136x processor features six synchronous serial ports that provide an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices. The serial ports can operate at up to one-fourth (0.25) of the processor peripheral clock rate with maximum of 41.625M bits per second. Each serial port features two data pins that function as a pair based on the same serial clock and frame sync. Accordingly, each serial port has two DMA channels and serial data buffers

associated with it to service the dual serial data pins. Programmable data direction provides greater flexibility for serial communications. Serial port data can automatically transfer to and from on-chip memory using DMA. Each of the serial ports offers a TDM multichannel mode (up to 128 channels) and supports µ-law or A-law companding. I^2S support is also provided.

The serial ports can operate with least significant bit first (LSBF) or most significant bit first (MSBF) transmission order, with word lengths from three to 32 bits. The serial ports offer selectable synchronization and transmit modes. Serial port clocks and frame syncs can be internally or externally generated.

Parallel port. The parallel port provides the processor interface to asynchronous 8-bit memory. The parallel port supports a 56M bytes per second transfer rate (PCLK/3) and 256 word page boundaries. The on-chip DMA controller automatically packs external data into the appropriate word width during transfers.

The parallel port supports packing of 32-bit words into 8-bit or 16-bit external memory and programmable external data access duration from 3 to 32 clock cycles.

Serial peripheral (compatible) interface (SPI). The SPI is an industry-standard synchronous serial link that enables the SPI-compatible port to communicate with other SPI-compatible devices. SPI is an interface consisting of two data pins, one device select pin, and one clock pin. It is a full-duplex synchronous serial interface, supporting both master and slave modes. It can operate in a multi-master environment by interfacing with up to four other SPI-compatible devices, either acting as a master or slave device.

The SPI-compatible peripheral implementation also supports programmable baud rate and clock phase/polarities, as well as the use of open drain drivers to support the multi-master scenario to avoid data contention.

ROM-based security. For those processors with application code in the on-chip ROM, an optional ROM securityfeature is included. This feature provides hardware support for securing user software code by preventing unauthorized reading from the enabled code. The processor does not boot-load any external code, executing exclusively from internal ROM. The processor also is not freely accessible via the JTAG port. Instead, a 64-bit key is assigned to the user. Thiskey must be scanned in through the JTAG or test access port (TAP). The device ignores a wrong key. Emulation features and external boot modes are only available after the correct key is scanned.

Digital Applications Interface (DAI)

The digital application interface (DAI) unit is an addition to the SHARC processor peripherals. This set of audio peripherals consists of an interrupt controller, an interface data port, a signal routing unit, two precision clock generators (PCGs), and three timers. Some family members have an S/PDIF receiver/transmitter, an asynchronous sample rate converter (SRC), and DTCP encryption.

Interrupt controller. The DAI contains its own interrupt controller that indicates to the core when DAI audioevents have occurred. This interrupt controller offer 32 independently configurable channels.

Input data port (IDP). The input data port provides the DAI with a way to transmit data from within the DAI to the core. The IDP provides a means for up to eight additional DMA paths from the DAI into on-chip memory. All eight channels support 32-bit wide data and share a 8-deep FIFO.

Signal routing unit (SRU). Conceptually similar to a “patch-bay” or multiplexer, the SRU provides a group of registers that define the interconnection of the serial ports, the input data port, the DAI pins, and the precision clock generators.

Development Tools

The ADSP-2136x processor is supported by VisualDSP++, an easy to use Integrated Development and Debugging Environment (IDDE). VisualDSP++ allows you to manage projects from start to finish from within a single, integrated interface. Because the project development and debug environments are integrated, you can move easily between editing, building, and debugging activities.

Architecture Enhancements

This section identifies differences between the ADSP-2136x processors and previous SHARC processors: ADSP-2116x, ADSP-2106x, and ADSP-21065L. Like the ADSP-2116x family, the ADSP-2136x processor family is based on the original ADSP-2106x SHARC family. The ADSP-2136x processor preserves much of the ADSP-2106x architecture and is code compatible to the ADSP-2116x, while extending performance and functionality. For background information on SHARC processors and the ADSP-2106x family processors, see the *ADSP-2106x SHARC User's Manual* or the *ADSP-21065L SHARC DSP Technical Reference*.

Parallel Port Enhancements

The parallel port includes a new packing mode which allows DMA for instructions and data to and from 8-bit external memory. The parallel port supports SRAM, EPROM, and flash memory. There are two modes supported for transfers. In one mode, 8-bit data and an 8-bit address can be transferred. In another mode, data and address lines are multiplexed to transfer 16 bits of address/data.

I/O Architecture Enhancements

The I/O processor provides greater throughput than the ADSP-2106x processors.

The DMA controller supports 25 channels compared to 14 channels on the ADSP-2116x processor.

Instruction Set Enhancements

The ADSP-2136x processor provides source code compatibility with the previous SHARC processor family members, to the assembly source code level. All instructions, control registers, and system resources available in the ADSP-2106x core programming model are also available with the ADSP-2136x processor. Instructions, control registers, or other facilities required to support the new peripherals of the ADSP-2136x core include:

- Code compatibility to the ADSP-21160 SIMD core
- Supersets of the ADSP-2106x programming model
- Reserved facilities in the ADSP-2106x programming model
- Symbol name changes from the ADSP-2106x and ADSP-2136x processor programming models

These name changes can be managed through reassembly by using the ADSP-2136x processor development tools to apply the ADSP-2136x processor symbol definitions header file and linker description file. While these changes have no direct impact on existing core applications, system and I/O processor initialization code and control code do require modifications.

Although the porting of source code written for the ADSP-2106x family to the ADSP-2136x processor has been simplified, code changes are required to take full advantage of the new ADSP-2136x processor features. For more information, see the *SHARC Processor Programming Reference*.

2 I/O PROCESSOR

In applications that use extensive off-chip data I/O, programs may find it beneficial to use a processor resource other than the processor core to perform data transfers. The ADSP-2136x prœssor contains an I/O processor (IOP) that supports a variety of DMA (direct memory access) operations. Each DMA operation transfers an entire block of data. These operations include the transfer types shown in Table 2-1 and the list that follows the table.

Table 2-1. I/O Processor Feature Summary

Feature	Availability
Total DMA channels	25
Rotating DMA channel priority	Yes
DMA channel interrupts	25
SPORT DMA channels	12
IDP DMA channel	8
SPI DMA channel	2
MTM DMA channel	2
PDAP DMA channel	1
Parallel Port DMA channel	1
Clock Operation	Peripheral clock (PCLK)

The I/O processor runs at CCLK ÷ 2 clock speed.

- Internal memory ↔ SPORT (DAI)
- Internal memory ← IDP (DAI) unidirectional
- Internal memory ↔ SPI
- Internal memory ↔ Internal memory (MTM)

By managing DMA, the I/O processor frees the processor core, allowing it to perform other operations while off-chip data I/O occurs as a background task. The multi-bank architecture of the ADSP-2136x internal memory allows the core and IOP to simultaneously access the internal memory if the accesses are to different memory banks. This means that DMA transfers to internal memory do not impact core performance. The processor core continues to perform computations without penalty.

To further increase off-chip I/O, multiple DMAs can occur at the same time. The IOP accomplishes this by managing multiple DMAs of processor memory through the different peripherals. Each DMA is referred to as a *channel* and each channel is configured independently. The IOP block diagram is shown in Figure 2-1 on page 2-15.

DMA Controller Operation

There are several methods used to run direct memory accesses on the SHARC processor.

Standard DMA. A standard DMA (once it is configured) transfers data from location A to location B. An interrupt can be used to indicate the end of the transfer. To start a new DMA sequence after the current one is finished, a program must first clear the DMA enable bit (control register), write new parameters to the index, modify, and count registers (parameter registers), then set the DMA enable bit to re-enable DMA (control register).

An instance where standard DMA can be used is to copy data from a peripheral to internal memory for processor booting. With the help of the loader tool, the tag (header information) of the boot stream is decoded to get the storage information which includes the index, modify, and count of a specific array to start another standard DMA.

Chained DMA. Chained DMA sequences are a set of multiple DMA operations, each autoinitializing the next in line. To start a new DMA sequence after the current one is finished, the IOP automatically loads new index, modify, and count values from an internal memory location (or external memory location for DMA to external ports) pointed to by that channel's chain pointer register. Using chaining, programs can set up consecutive DMA operations and each operation can have different attributes.

The IDP port does not support DMA chaining.

Ping-pong DMA (IDP). In ping-pong DMA, the parameters have two memory index values (index A and index B), one count value and one modifier value. The DMA starts the transfer with the memory indexed by A. When the transfer is completed as per the value in the count register, the DMA restarts with the memory location indexed by B. The DMA restarts with index A after the transfer to memory with index B is completed as per the count value. This repeats until the DMA is stopped by resetting the DMA enable bit.

DMA Transfers Between Internal Memory

The ADSP-2136x processors can use memory-to-memory DMA to transfer 64-bit blocks of data between internal memory locations.

General Procedure for Configuring DMA

To configure the processors to use DMA, use the following general procedure.

1. Configure the peripheral(s):
    - Parallel port (PPCTL)
    - Serial ports (SPORTs)
    - Serial peripheral interface ports (SPI)
    - Input data port (IDP)
2. Determine which DMA options you want to use:
    - DMA transfer method – chained or non chained
    - Select DMA channels (for example, the SPORT has 12 channels
3. Determine the DMA channel priority
    - Channel priority scheme – fixed or rotating
4. Determine the address region in which you want the DMA to operate
    - Set up the data's source/destination addresses (INDEX)
    - Set up the word COUNT (data buffer size)
    - Configure the MODIFY values (step size)
5. Set the DMA interaction method
    - IOP/core interaction method – interrupt-driven or status-driven (polling)

6. Start the DMA
    - Set the applicable bits in the appropriate registers.

The following sections provide more detailed DMA information for specific peripherals.

Summary

Because the IOP registers are memory-mapped, the processors have access to program DMA operations. A program sets up a DMA channel by writing the transfer's parameters to the DMA parameter registers. After the index, modify, and count registers (among others) are loaded with a starting source or destination address, an address modifier, and a word count, the processor is ready to start the DMA.

The peripherals each have a DMA enable (`xDEN`) bits in their channel control registers. Setting this bit for a DMA channel with configured DMA parameters starts the DMA on that channel. If the parameters configure the channel to receive, the I/O processor transfers data words received at the buffer to the destination in internal memory. If the parameters configure the channel to transmit, the I/O processor transfers a word automatically from the source memory to the channel's buffer register. These transfers continue until the I/O processor transfers the selected number of words as determined by the count parameter. DMA through the IDP ports occurs in internal memory only.

IOP Registers

The IOP registers are memory mapped and can be accessed through an address by the core. For the list of parameter registers refer to “Registers Reference” in Appendix A, Registers Reference.

Standard DMA Parameter Registers

The parameter registers described below control the source and destination of the data, the size of the data buffer, and the step size used.

Index registers. Shown in Table 2-2, provide an internal memory address, acting as a pointer to the next internal memory DMA read or write location. All internal index addresses are based on an internal memory offset of 0x80000.

Table 2-2. Index Registers

Register Name	Width (Bits)	Description
IISP0–5A	19	SPORTA
IISP0–5B	19	SPORTB
IISPI	19	SPI
IISPIB	19	SPIB
IDP_DMA_I0–7	19	IDP
IDP_DMA_I0–7A	19	IDP index A (ping pong)
IDP_DMA_I0–7B	19	IDP index B (ping pong)
IIMTMW	19	MTM Write
IIMTMR	19	MTM Read
IIPP	19	Parallel Port
EIPP	24	Parallel Port (external address)

Modify registers. Shown in Table 2-3, provide the signed increment by which the DMA controller post-modifies the corresponding memory index register after the DMA read or write.

Table 2-3. Modify Registers

Register Name	Width (Bits)	Description
IMSP0–5A	16	SPORTA
IMSP0–5B	16	SPORTB
IMSPI	16	SPI
IMSPIB	16	SPIB
IDP_DMA_M0–7	6	IDP
IDP_DMA_M0–7A	6	IDP modify A (ping pong)
IDP_DMA_M0–7B	6	IDP modify B (ping pong)
IMMTMW	16	MTM Write
IMMTMR	16	MTM Read
IMPP	16	Parallel Port
EMPP	2	Parallel Port (external address)

Count registers. Shown in Table 2-4, indicate the number of words remaining to be transferred to or from memory on the corresponding DMA channel.

Table 2-4. Count Registers

Register Name	Width (Bits)	Description
ICSP0–5A	16	SPORTA
ICSP0–5B	16	SPORTB
ICSPI	16	SPI
ICSPIB	16	SPIB
IDP_DMA_C0–7	16	IDP
ICMTMW	16	MTM Write
ICMTMR	16	MTM Read
ICPP	16	Parallel Port
ECPP	16	Parallel Port (external address)

Chain pointer registers. Shown in Table 2-5, hold the starting address of the TCB (parameter register values) for the next DMA operation on the corresponding channel. These registers also control whether the I/O processor generates an interrupt when the current DMA process ends.

Table 2-5. Chain Pointer Registers

Register Name	Width (Bits)	Description
CPSP0–5A	29	SPORTA
CPSP0–5B	29	SPORTB
CPSPI	20	SPI
CPSPIB	20	SPIB
CPPP	20	Parallel Port

Standard DMA Status Registers

The registers shown in Table 2-6 provide information on the standard DMA status registers.

Table 2-6. Standard DMA Status Registers

Register Name	Width (Bits)	Description
PPCTL	32	Parallel port control register
SPMCTLxy	32	SPORT Multichannel Control Registers
SPIDMAC	32	SPI DMA control
SPIDMACB	32	SPIB DMA control

Chaining DMA Status Registers

In the registers shown in Table 2-7 Two status bits are available: one for DMA status and one for chain loading DMA status.

Table 2-7. Chaining DMA Status Registers

Register Name	Width (Bits)	Description
PPCTL	32	Parallel port control
SPMCTLxy	32	SPORT Multichannel control
SPIDMAC	32	SPI DMA control
SPIDMACB	32	SPIB DMA control
MTMCTL	32	MTM DMA control

Data Buffers

The data buffers or FIFOs are used by each DMA channel to store data during the priority arbitration time period (shown in Table 2-8). The buffers (depending on the peripheral) are accessed by both DMA and the core. The data buffers are described in the following list.

Table 2-8. Data Buffers

Buffer Name	FIFO Depth	Description
TXSP0–5A	2	SPORTA Transmit
TXSP0–5B	2	SPORTB Transmit
RXSP0–5A	2	SPORTA Receive
RXSP0–5B	2	SPORTB Receive
TXSPI	2	SPI Transmit
TXSPIB	2	SPIB Transmit
RXSPI	2	SPI Receive
RXSPIB	2	SPIB Receive

Table 2-8. Data Buffers (Cont'd)

Buffer Name	FIFO Depth	Description
SPI DMA	4	(DMA only)
SPIB DMA	4	(DMA only)
IDP_FIFO	8	IDP FIFO Receive
MTM read/write	2	DMA only
TXPP	2	Parallel Port Transmit
RXPP	2	Parallel Port Receive

Some data buffers provide debug support to enable the buffer hang disable (BHD) bit. This feature can be enabled in the dedicated peripheral control register for the IDP and SPORT.

DMA Channel Allocation

There are 25 channels of DMA available on the ADSP-2136x processors depending on the processor model. The DMA channels are configured as follows.

- 12 via the serial ports (channels 0–5/A and B)
- 8 via the input data port IDP (channels 0–7)
- 2 via SPI/B (one each)
- 2 via MTM (one each for read and write)
- 1 via parallel port

Each channel has a set of parameter registers which are used to set up DMA transfers. Table 2-9 shows the DMA channel allocation and parameter register assignments for the ADSP-2136x processors.

DMA channels vary by processor model. For a breakdown of DMA channels for a particular model, see the *ADSP-2136x SHARC Processor Data Sheet*. Also note that each DMA channel has a specific peripheral assigned to it.

DMA Channel Priority

Table 2-9 shows the paths for internal DMA requests within the I/O processor.

Table 2-9. DMA Channel Priorities

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
0 (Highest Priority	A	SPCTL0, SPMCTL01	IISP0A, IMSP0A, CSP0A, CPSP0A	RXSP0A or TXSP0A	Serial Port 0A Data
1		SPCTL0, SPMCTL01	IISP0B, IMSP0B, CSP0B, CPSP0B	RXSP0B or TXSP0B	Serial Port 0B Data
2		SPCTL1, SPMCTL01	IISP1A, IMSP1A, CSP1A, CPSP1A	RXSP1A or TXSP1A	Serial Port 1A Data
3		SPCTL1, SPMCTL01	IISP1B, IMSP1B, CSP1B, CPSP1B	RXSP1B or TXSP1B	Serial Port 1B Data
4	B	SPCTL2, SPMCTL23	IISP2A, IMSP2A, CSP2A, CPSP2A	RXSP2A or TXSP2A	Serial Port 2A Data
5		SPCTL2, SPMCTL23	IISP2B, IMSP2B, CSP2B, CPSP2B	RXSP2B or TXSP2B	Serial Port 2B Data
6		SPCTL3, SPMCTL23	IISP3A, IMSP3A, CSP3A, CPSP3A	RXSP3A or TXSP3A	Serial Port 3A Data
7		SPCTL3, SPMCTL23	IISP3B, IMSP3B, CSP3B, CPSP3B	RXSP3B or TXSP3B	Serial Port 3B Data

Table 2-9. DMA Channel Priorities (Cont'd)

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
8	C	SPCTL4, SPMCTL45	IISP4A, IMSP4A, CSP4A, CPSP4A	RXSP4A or TXSP4A	Serial Port 4A Data
9		SPCTL4, SPMCTL45	IISP4B, IMSP4B, CSP4B, CPSP4B	RXSP4B or TXSP4B	Serial Port 4B Data
10		SPCTL5, SPMCTL45	IISP5A, IMSP5A, CSP5A, CPSP5A	RXSP5A or TXSP5A	Serial Port 5A Data
11		SPCTL5, SPMCTL45	IISP5B, IMSP5B, CSP5B, CPSP5B	RXSP5B or TXSP5B	Serial Port 5B Data
12	D	IDP_CTL0, IDP_CTL1, IDP_FIFO, IDP_PP_CTL, DAI_STAT	IDP_DMA_I0, IDP_DMA_M0, IDP_DMA_C0, IDP_DMA_I0A, IDP_DMA_I0B, IDP_DMA_PC0	IDP_FIFO	DAI IDP or PDAP (only channel 0 supports both
13		IDP_CTL0, IDP_CTL1, IDP_FIFO, DAI_STAT	IDP_DMA_I1, IDP_DMA_M1, IDP_DMA_C1, IDP_DMA_I1A, IDP_DMA_I1B, IDP_DMA_PC1	IDP_FIFO	DAI IDP Channel 1
14		IDP_CTL0, IDP_CTL1, IDP_FIFO, DAI_STAT	IDP_DMA_I2, IDP_DMA_M2, IDP_DMA_C2, IDP_DMA_I2A, IDP_DMA_I2B, IDP_DMA_PC2	IDP_FIFO	DAI IDP Channel 2
15		IDP_CTL0, IDP_CTL1, IDP_FIFO, DAI_STAT	IDP_DMA_I3, IDP_DMA_M3, IDP_DMA_C3, IDP_DMA_I3A, IDP_DMA_I3B, IDP_DMA_PC3	IDP_FIFO	DAI IDP Channel 3

Table 2-9. DMA Channel Priorities (Cont'd)

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
16	D	IDP_CTL0, IDP_CTL1, IDP_FIFO, DAI_STAT	IDP_DMA_I4, IDP_DMA_M4, IDP_DMA_C4, IDP_DMA_I4A, IDP_DMA_I4B, IDP_DMA_PC4	IDP_FIFO	DAI IDP Channel 4
17		IDP_CTL0, IDP_CTL1, IDP_FIFO, DAI_STAT	IDP_DMA_I5, IDP_DMA_M5, IDP_DMA_C5, IDP_DMA_I5A, IDP_DMA_I5B, IDP_DMA_PC5	IDP_FIFO	DAI IDP Channel 5
18		IDP_CTL0, IDP_CTL1, IDP_FIFO, DAI_STAT	IDP_DMA_I6, IDP_DMA_M6, IDP_DMA_C6, IDP_DMA_I6A, IDP_DMA_I6B, IDP_DMA_PC6	IDP_FIFO	DAI IDP Channel 6
19		IDP_CTL0, IDP_CTL1, IDP_FIFO, DAI_STAT	IDP_DMA_I7, IDP_DMA_M7, IDP_DMA_C7, IDP_DMA_I7A, IDP_DMA_I7B, IDP_DMA_PC7	IDP_FIFO	DAI IDP Channel 7
20	E	SPICTL, SPIDMAC, SPIBAUD SPISTAT	IISPI, IMSPI, CSPI, CPSPI	RXSPI or TXSPI	SPI Data
21	F	PPCTL	IIPP, IMPP, ICPP, EIPP, ECPP, EMPP, CPPP	RXPP or TXPP	Parallel Port Data
22	G	SPICTLB, SPIDMACB, SPIBAUDB, SPISTATB	IISPIB, IMSPIB, CSPIB, CPSPIB	RXSPIB or TXSPIB	SPI B Data

Table 2-9. DMA Channel Priorities (Cont'd)

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
23	H	MTMCTL	IIMTMW, IMMTMW, CMTMW	MTM FIFO	Memory-to-memory write data
24	I	MTMCTL	IIMTMR, IMMTMR, CMTMR	MTM FIFO	Memory-to-memory read data

DMA Channel Arbitration Modes

DMA channel arbitration is the method that the arbiter uses to determine how groups rotate priority with other channels. The default DMA channel priority is fixed prioritization by DMA channel group.

Peripheral DMA Bus

DMA-capable peripherals execute DMA data transfers to and from internal memory over the IOD bus (Figure 2-1). When more than one of these peripherals requests access to the IOD bus in a clockcycle, the bus arbiter, which is attached to the IOD bus, determines which master should have access to the bus and grants the bus to that master.

The core user breakpoint unit allows monitoring of the IOD bus activities. For more information, refer to the JTAG chapter in the *SHARC Processor Programming Reference*.

IOP channel arbitration can be set to use either a fixed or rotating algorithm by setting or clearing `DCPR` bit in the `SYSCTL` register as follows.

- (=0) fixed arbitration (default)
- (=1) rotating arbitration

In the fixed priority scheme, the lower indexed peripheral (Table 2-9) has the highest priority.

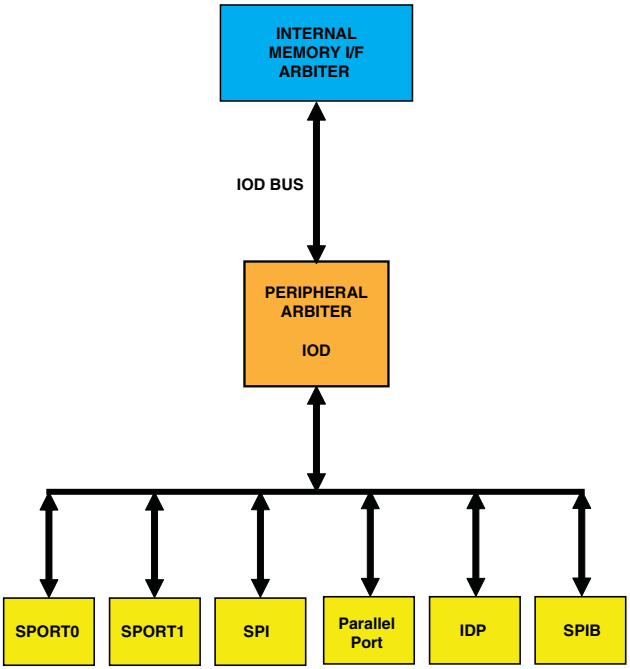


Figure 2-1. I/O Processor Bus Structure

Rotating Priority by Group

In the rotating priority scheme, the default priorities at reset are the same as that of the fixed priority. However, the peripheral priority is determined by group, not individually by DMA channel. Peripheral groups are shown in Table 2-9 on page 2-11.

Initially, group A has the highest priority and group I the lowest. As one group completes its DMA operation, it is assigned the lowest priority (moves to the back of the line) and the next group is given the highest priority.

When none of the peripherals request bus access, the highest priority peripheral, for example, peripheral 0, is granted the bus. However, this does not change the currently assigned priorities to various peripherals.

Within a peripheral group, the priority is highest for the higher indexed peripheral (see Table 2-9). For example, the SPORT pair SP01 (which is in group A), SP1 has the highest priority. Changes of DMA arbitration modes between fixed and rotate can be done on the fly which has effect latency of 2 PCLK cycles.

DMA Channel Interrupts

The next two sections describe the types of DMA interrupts based on the peripheral that is used.

Internal Transfer Completion

This mode of interrupt generation resembles the traditional SHARC DMA interrupt generation. The interrupt is generated once the DMA internal transfers are done independent of whether it is transmit or receive DMA, hence for external transmit DMA case when the completion interrupt is generated there may be still external access pending at the external DMA interface.

DMA Channel Interrupt Priorities

Interrupts are generated at the end of a DMA transfer. This happens when the count register for a particular channel decrements to zero. The default interrupt vector locations for each of the channels are listed in "Interrupts" in Appendix B, Interrupts and "DAI Interrupt Controller Registers" on page A-77.

The digital applications interface (DAI) has two interrupts—a lower priority option (DAILI) and a higher priority option (DAIHI). This allows

two interrupts to have priorities that are higher and lower than the serial ports.

For more information, see the Program Sequencer "Interrupts and Sequencing" in the *SHARC Processor Programming Reference*.

Interrupt Versus Channel Priorities

At default, the DMA interrupt priorities do not match the DMA channel priorities (Table 2-10). However, if both priorities schemes should match or should change, the DMA interrupt priorities can be re-assigned by dedicated settings of the `PICRx` registers.

Table 2-10. Default DMA Channel versus Interrupt Priorities

Programmable Interrupt	Default Interrupt Priority	Priorities	DMA Channel Priority
P0I	DAIHI	Highest	SPORT[5–0] – 12 channels
P1I	SPII		
P3I	SP1I		IDP[7–0] – 8 channels
P4I	SP3I		
P5I	SP5I		SPI – 1 channel
P6I	SP0I		
P7I	SP2I		PP – 1 channel
P8I	SP4I		
P9I	PPI		SPI B – 1 channel
P12I	DAILI		
P15I	MTMI		MTM [WR/RD] – 2 channels
P18I	SPIBI	Lowest	

The `PICRx` (peripheral interrupt priority control registers) allow a change of the default interrupt priorities. For more information, see "Programmable Interrupt Control Registers" on page B-1.

DMA Controller Addressing

Figure 2-2 shows a block diagram of the I/O processor's address generator (DMA controller). "Standard DMA Parameter Registers" on page 2-6 lists the parameter registers for each DMA channel. The parameter registers are uninitialized following a processor reset.

The I/O processor generates addresses for DMA channels much the same way that the Data Address Generators (DAGs) generate addresses for data memory accesses. Each channel has a set of parameter registers including an index register and modify registerthat the I/O processor uses to address a data buffer in internal memory. The index register must be initialized with a starting address for the data buffer. As part of the DMA operation, the I/O processor outputs the address in the index register onto the processor's I/O address bus and applies the address to internal memory during each DMA cycle—a clock cycle in which a DMA transfer is taking place.

Internal Index Register Addressing

All addresses in the index registers are offset by a value matching the processor's first internal normal word addressed RAM location, before the I/O processor uses the addresses. For the ADSP-2136x processors, this offset value is 0x0008 0000.

The following rules for data transfers must be followed.

- DMA index addresses must always be normal word space (32-bit).
- The I/O processor can transfer short word data (16-bit or 8-bit) using the packing capability of the peripherals (serial port or SPI). The data are packed in the peripheral's shift register to form 32-bit words for the internal transfers over the IOD bus.

After transferring each data word to or from internal memory, the I/O processor adds the modify value to the index register to generate the

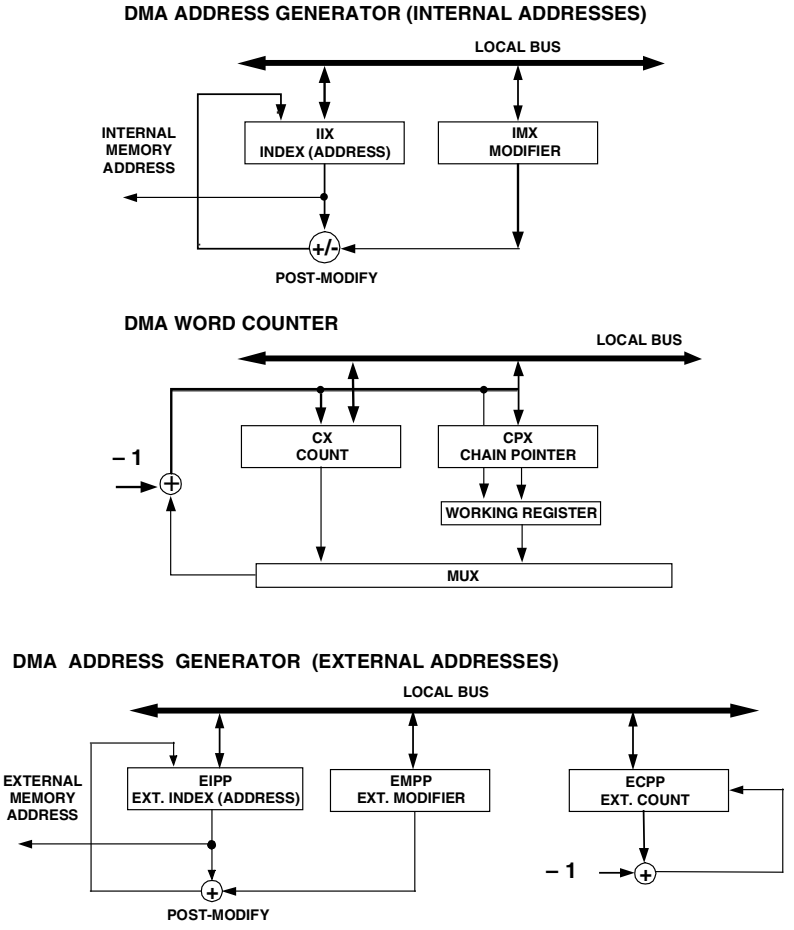


Figure 2-2. DMA Address Generator

address for the next DMA transfer and writes the modified index value to the index register. The modify value in the modify register is a signed integer, which allows both increment and decrement modifies. The modify value can have any positive or negative integer value. Note that:

- If the I/O processor modifies the internal index register past the maximum 19-bit value to indicate an address out of internal memory, the index wraps around to zero. With the offset for the SHARC processor, the wraparound address is 0x80000.
- If a DMA channel is disabled, the I/O processor does not service requests for that channel, whether or not the channel has data to transfer.

If a program loads the count register with zero, the I/O processor does not disable DMA transfers on that channel. The I/O processor interprets the zero as a request for 2^{16} transfers. This count occurs because the I/O processor starts the first transfer before testing the count value. The only way to disable a DMA channel is to clear its DMA enable bit.

DMA Chaining

In many systems, some types of data transfers are continuous or periodic. In order to implement these data transfers without any core intervention, automation can be used. The chained DMA mode helps to accomplish this automation. With chained DMA, the attributes of a specific DMA are stored in internal memory and are referred to as a *Transfer Control Block* or TCB. The DMA controller loads these attributes in chains for execution. This allows for multiple chains that are an finite or infinite.

If chaining is enabled on a DMA channel, programs should not use polling to determine channel status as this gives inaccurate information where the DMA appears inactive if it is sampled while the next TCB is loading.

TCB Memory Storage

The location of the DMA parameters for the next sequence comes from the chain pointer register. In chained DMA operations, the processor

automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. In addition to the standard DMA parameter registers, each DMA channel also has a chain pointer register that points to the next set of DMA parameters stored in the processor's internal memory. Each new set of parameters is stored in a user-initialized memory buffer (Table 2-11) known as a transfer control block (TCB)

Table 2-11. Principal TCB Allocation for a Serial Peripheral

Address	Register	Description
CPx – 0x3 (IIx)	Internal index register	Internal memory buffer
CPx – 0x2 (IMx)	Internal modify register	Stride for internal buffer
CPx – 0x1 (ICx)	Internal count register	Length of internal buffer
CPx	Chain pointer register	Chain pointer for DMA chaining

The size of TCB varies and is based on the peripheral to be used: the SPORTs, and SPI require 4 locations. Different TCB sizes are advantageous since only the required TCBs are allocated in internal memory, which reduces the memory load.

Chain Pointer Register

The chain pointer register, described in Table 2-12 is 20 bits wide. The lower 19 bits are the memory address field. Like other I/O processor address registers, the chain pointer register's value is offset to match the starting address of the processor's internal memory before it is used by the I/O processor. On the SHARC processor, this offset value is 0x80000.

Table 2-12. Chain Pointer Register for SPORTs, SPI, Parallel Port (xCPx)

Bit	Name	Description
18–0	IIx address	Next chain pointer address
19	PCI	Program controlled interrupt 0 = no interrupt after current TCB 1 = interrupt after current TCB

Bit 19 of the chain pointer register is the program controlled interrupt (PCI) bit. This bit controls whether an interrupt is latched after every DMA in the chain (when set = 1), or whether the interrupt is latched after the entire DMA sequence completes (if cleared = 0).

The PCI bit only effects DMA channels that have chaining enabled. Also, interrupt requests enabled by the PCI bit are maskable with the IMASK register.

Chain Assignment

The structure of a TCB is conceptually the same as that of a traditional linked list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB may point to itself to continuously re run the same DMA. The I/O processor reads each word of the TCB and loads it into the corresponding register.

Programs must assign the TCB in memory in the order shown in Figure 2-3, placing the index parameter at the address pointed to by the chain pointer register of the previous DMA operation of the chain. The end of the chain (no further TCBs are loaded) is indicated by a TCB with a chain pointer register value of zero.

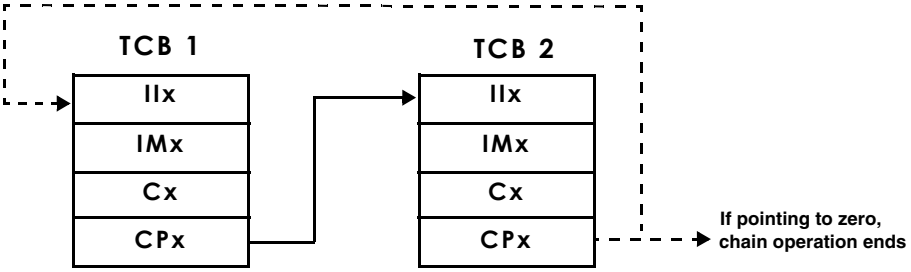


Figure 2-3. Chaining in the SPI and Serial Ports

The address field of the chain pointer registers is only 19 bits wide. If a program writes a symbolic address to bit 19 of the chain pointer there may be a conflict with the PCI bit. Programs should clear the upper bits of the address then AND the PCI bit separately, if needed, as shown below.

Clear the chain pointer register before chaining is enabled.

```
Chain Assignment (according to Figure 2-3):
R0=0;
dm(CPx)=R0;            /* clear CPx register */

/* init DMA control registers */

R2=(TCB1+3) & 0x7FFFF;    /* load IIx address of next TCB
                             and mask address */
R2=bset R2 by 19;        /* set PCI bit */
dm(TCB2)=R2;             /* write address to CPx location of
                            current TCB */
R2=(TCB2+3) & 0x7FFFF;   /* load IIx address of next TCB and
                             mask address*/
R2=bclr R2 by 19;         /* clear PCI bit */
dm(TCB1)=R2;            /* write address to CPx location of
                            current TCB */

dm(CPx)=R2;             /* write IIx address of TCB1 to CPx
                             register to start chaining*/
```

Chained DMA operations may only occur within the same channel. The processor does not support cross-channel chaining.

Starting Chain Loading

A DMA sequence is defined as the sum of the DMA transfers for a single channel, from when the parameter registers initialize to when the count register decrements to zero. Each DMA channel has a chaining enable bit (CHEN) in the corresponding control register.

To start the chain, write the internal index address of the first TCB to the chain pointer register. When chaining is enabled, DMA transfers are initiated by writing a memory address to the chain pointer register. This is also an easy way to start a single DMA sequence, with no subsequent chained DMAs.

During TCB chain loading, the I/O processor loads the DMA channel parameter registers with values retrieved from internal memory. Note that the SPI Port is one exception. To execute the first DMA in a chain for this peripheral, the DMA parameter registers also need to be explicitly programmed. For more information, see “Serial Peripheral Interface Ports” in Chapter 7, Serial Peripheral Interface Ports.

The address in the chain pointer register points to the highest address of the TCB (containing the index parameter). This means that if a program declares an array to hold the TCB, the chain pointer register should point to the last location of the array and not to the first TCB location.

The chain pointer register can be loaded at any time during the DMA sequence. This allows a DMA channel to have chaining disabled (chain pointer register address field = 0x0) until some event occurs that loads the chain pointer register with a non zero value. Writing all zeros to the address field of the chain pointer register also disables chaining.

TCB Chain Loading Priority

A TCB chain load request is prioritized like all DMA channels. Therefore, the TCB chain loading request has the same priority level as the DMA channel itself. The I/O processor latches a TCB loading request and holds it until the load request has the highest priority. If multiple chaining requests are present, the I/O processor services the TCB block for the highest priority DMA channel first.

A channel that is in the process of chain loading cannot be interrupted by any other request (TCB, DMA channel). The chain loading sequence is atomic and the I/O bus is locked until all the DMA parameter registers are loaded. For a list of DMA channels in priority order, see Table 2-9.

Chain Insert Mode (SPORTs Only)

It is possible to insert a single SPORT DMA operation or another DMA chain within an active SPORT DMA chain. Programs may need to perform insertion when a high priority DMA requires service and cannot wait for the current chain to finish. This is supported only for SPORT DMA channels only. For more information, see "Serial Ports" in Chapter 6, Serial Ports.

DMA Start and Stop Conditions

The difference between single DMA and chained DMA is based on the auto-linkage process where the DMA's attributes are stored in internal memory and automatically loaded by the IOP if requested.

A DMA sequence starts when one of the following occurs.

- Chaining is disabled, and the DMA enable bit transitions from low to high.
- Chaining is enabled, DMA is enabled, and the chain pointer register address field is written with a non zero value. In this case, TCB chain loading of the channel parameter registers occurs first.
- Chaining is enabled, the chain pointer register address field is non-zero, and the current DMA sequence finishes. Again, TCB chain loading occurs.

A DMA sequence ends when one of the following occurs.

- The count register decrements to zero, and the chain pointer register is zero.
- Chaining is disabled and the channel's DMA enable bit transitions from high to low. If the DMA enable bit goes low (=0) and chaining is enabled, the channel enters chain insertion mode (SPORT only) and the DMA sequence continues.

Once a program starts a DMA process, the process is influenced by two external controls—DMA channel priority and DMA chaining.

Configuring IOP/Core Interaction

There are two methods the processor uses to monitor the progress of DMA operations—interrupts, which are the primary method, and status polling. The same program can use either method for each DMA channel. The following sections describe both methods in detail.

Interrupt-Driven I/O

Programs can check the appropriate status register (for example SPCTL for the serial ports) to determine which channels are performing a DMA or chained DMA.

All DMA channels can be active or inactive. If a channel is active, a DMA is in progress on that channel. The I/O processor indicates the active status by setting the channel's bit in the status register. The only exception to this is the IDP_DMAx_STAT bits of the DAI_STAT register can become active even if DMA, through some IDP channel, is not intended.

The following are some other I/O processor interrupt attributes.

- When an unchained (single block) DMA process reaches completion (as the count decrements to zero) on any DMA channel, the I/O processor latches that DMA channel's interrupt. It does this by setting the DMA channel's interrupt latch bit in the IRPTL, LIRPTL, DAI_IRPTL_H, or DAI_IRPTL_L registers.
- For chained DMA, the I/O processor generates interrupts in one of two ways:
  If PCI = 1, (bit 19 of the chain pointer register is the program controlled interrupts (PCI) bit) an interrupt occurs for each DMA in the chain.
  If PCI = 0, an interrupt occurs at the end of a complete chain. (For more information on DMA chaining, see "DMA Controller Operation" on page 2-2).
- When a DMA channel's buffer is not being used for a DMA process, the I/O processor can generate an interrupt on single word writes or reads of the buffer. This interrupt service differs slightly for each port.

During interrupt-driven DMA, programs use the interrupt mask bits in the IMASK, LIRPTL, DAI_IRPTL_PRI, DAI_IRPTL_RE, and DAI_IRPTL_FE registers to selectively mask DMA channel interrupts that the I/O

processor latches into the IRPTL, LIRPTL, DAI_IRPTL_H, and DAI_IRPTL_L registers.

The I/O processor only generates a DMA complete interrupt when the channel's count register decrements to zero as a result of actual DMA transfers. Writing zero to a count register does not generate the interrupt. To stop a DMA preemptively, write a one to the count register. This causes one additional word to be transferred or received, and an interrupt is then generated.

A channel interrupt mask in the IMASK, LIRPTL, DAI_IRPTL_PRI, DAI_IRPTL_RE, and DAI_IRPTL_FE registers determines whether a latched interrupt is serviced or not. When an interrupt is masked, it is latched but not serviced. For more information, see the "Registers" section of the *SHARC Processor Programming Reference*.

By clearing a channel's PCI bit during chained DMA, programs mask the DMA complete interrupt for a DMA process within a chained DMA sequence.

The I/O processor can also generate interrupts for I/O port operations that do not use DMA. In this case, the I/O processor generates an interrupt when data becomes available at the receive buffer or when the transmit buffer is not full (when there is room for the core to write to the buffer). Generating interrupts in this manner lets programs implement interrupt-driven I/O under control of the processor core. Care is needed because multiple interrupts can occur if several I/O ports transmit or receive data in the same cycle.

Polling DMA Channel Status

The second method of controlling I/O is through status polling. The DMA controllers in the ADSP-2136x processor maintain the status information of each channel in each of the peripherals registers.

Because polling uses processor core resources, it is not as efficient as an interrupt-driven system. Also note that polling the DMA status registers reduces I/O bandwidth (core higher priority like I/O, see "IOP Performance" on page 2-35).

The DMA controllers in the ADSP-2136x processor maintain the status information of each channel for the different DMA modes in each of the peripherals registers:

- PPCTL (Standard and chaining)
- SPMCTLxy (Standard and chaining)
- SPIDMAC, SPIDMACB (Standard and chaining)
- DAI_STAT (Standard, Ping-pong)
- MTMCTL (Standard)

Note that there is a one cycle latency between a change in DMA channel status and the status update in the corresponding register.

Standard DMA Status

By starting DMA, the related DMA status bit is set and cleared at the end of the DMA.

Chaining DMA Status

Two status bits are available for chaining status—one bit for DMA status and one bit for chain loading DMA status. When a program starts a chained DMA, chain loading is triggered, and the related chain loading status bit is set. After the TCB loading completes, its status bit is cleared and the DMA status bit is set. This scenario is repeated until the chain pointer reaches zero. Note that there is a one cycle latency between a change in DMA channel status and the status update in the corresponding register.

TCB Storage

This section lists all the different TCB memory allocations used for DMA chaining on the peripherals. Note that all TCBs must be located in internal memory.

Serial Port TCB

The serial ports support single and chained DMA. Table 2-13 shows the required TCB for chained DMA

Table 2-13. SPORT TCBs

Address	Register
CP[18:0]	IISPx Internal/External Index
CP[18:0] – 0x1	IMSPx Internal Modifier
CP[18:0] – 0x2	ICSPx Internal Count
CP[18:0] – 0x3	CPSPx Chain Pointer

Parallel Port TCB

The parallel port interface supports both single and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain. Table 2-14 shows the required TCB for chained DMA.

Table 2-14. Parallel Port TCBs

Address	Register
CP[18:0]	IIPP Internal Index
CP[18:0] – 0x1	IMPP Internal Modifier
CP[18:0] – 0x2	ICPP Internal Count
CP[18:0] – 0x3	CPPP Chain Pointer

Table 2-14. Parallel Port TCBs (Cont'd)

Address	Register
CP[18:0] – 0x4	EIPP External Index
CP[18:0] – 0x5	EMPP External Modifier
CP[18:0] – 0x6	ECPP External Count

For more information on programming DMA, refer to the specific peripheral chapters.

For the parallel port chain pointer, the TCB location is not at the beginning of the TCB list.

SPI TCB

The serial peripheral interfaces supports both single and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain. Table 2-15 shows the required TCB for chained DMA.

Table 2-15. SPI/SPIB TCBs

Address	Register
CP[18:0]	IISPI/B Internal Index
CP[18:0] – 0x1	IMSPI/B Internal Modifier
CP[18:0] – 0x2	ICSPI/B Internal Count
CP[18:0] – 0x3	CPSPI/B Chain Pointer

I/O Processor Register Access

All of the I/O processor's registers are memory-mapped, ranging from address 0x0000 0000 to 0x0003 FFFF.

IOP Access Conditions

The IOP registers are physically located in two clock domains.

1. Core domain (CCLK)—SYSCTL and all user breakpoint registers. For more information, see "Registers Reference" in Chapter A, Registers Reference.
2. Peripheral domain (PCLK)—All other IOP registers are located in the peripheral domain (core/2 clock). This means that there are different access conditions which are explained in this section.

Accesses to IOP registers (from the processor core) should not use Type 1 (dual access) or LW instructions.

I/O processor registers have an effect latency range from one to six cycles (changes take minimum effect on the second cycle or maximum effect after seven cycles).

Table 2-16. I/O Processor Access Conditions

Type Of Access	Core Domain (Core Cycles)	Peripheral Domain (Core Cycles)
IOP register write/read	1/2	1/8
IOP register back to back write/read	1/2	2/8
Conditional IOP register write/read	1/2	3/10
Aborted IOP register write/read	2/3	4/4

The following situations also incur additional stall cycles.

1. Attempting to write to (or read from) a full (or empty) DMA buffer (IDP and SPORT) causes the core to hang indefinitely, unless the BHD (buffer hang disable) bit for that peripheral is set (SPCTLx, PPCTL, IDP_CTL).
2. In case of a full write data FIFO, the held-offI/O processor register read or write access incurs one extra core-clock cycle.
3. Interrupted IOP register reads and writes, if preceded by another write creates one additional core stall cycle.

Interrupt Latency

During an interrupt-driven I/O transfer from any peripheral that uses an IOP interrupt service routine, a write into an IOP register to clear the interrupt causes a certain amount of latency. If the program comes out of the interrupt service routine during that period of latency, the interrupt is generated again.

To avoid the interrupt from being regenerated, use one of the following solutions.

1. Read an IOP register from the same peripheral block before the return from interrupt (RTI). The read forces the write to occur as shown in the example code below.

```
ISR_Routine:
     R0 = 0x0;
     dm(SPICTL) = R0;  /* disable SPI */
     R0 = dm(SPICTL);  /* dummy read, occurs only after
                       write */
     rti;
```

2. Add sufficient NOP instructions after a write. In the worst case programs need to add ten NOP instructions after a write as shown in the example code below.

```
ISR_Routine:
     R0 = 0x0;
     dm(SPICTL) = R0;  /* disable SPI */
     lcntr=10, do (pc,1) until lce;

     nop;
rti;
```

TCB Chain Loading Access

Table 2-17 lists the time required to load a specific TCB from the internal memory into the DMA controller. During this time, the IOD bus is locked and cannot be interrupted.

IOP Register Access Arbitration

Since the I/O processor's registers arepart of the processor's memory map, buses access these registers as locations in memory. While these registers act as memory-mapped locations, they are separate from the processor's internal memory and have different bus access. One bus can access one I/O processor register at a time. When there is contention among the buses for access to the same I/O processor register, the processor arbitrates register access as follows.

1. DMD bus accesses (highest priority)
2. PMD bus accesses
3. IOD bus accesses (lowest priority)

Note that internal memory block access arbitration is different—the highest priority is in favor of the IOD followed by the DMD and finally the PMD buses.

IOP Performance

Since the I/O processor controls the I/O bus, the maximum bandwidth is achieved with PCLK × 32-bit as shown in Table 2-17.

Table 2-17. I/O Processor TCB Chain Loading Access

Chained TCB Type	TCB Size	Number of Core Cycles
SPI DMA, SPORT DMA	4	26
Parallel Port DMA	7	40

3 MEMORY-TO-MEMORY PORT DMA

Table 3-1 and the following list describe the MTM features.

Table 3-1. MTM Port Feature Summary

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
Interrupt Default Routing	Yes (P15I)
Protocol	
Master Capable	Yes
Slave Capable	No
Transmission Simplex	Yes
Transmission Half Duplex	No
Transmission Full Duplex	No
Access Type	
Data Buffer	Yes
Core Data Access	No
DMA Data Access	Yes
DMA Channels	2
DMA Chaining	No
Interrupt Source	DMA
Boot Capable	No

Table 3-1. MTM Port Feature Summary (Cont'd)

Feature	Availability
Local Memory	No
Clock Operation	PCLK

Features

The memory-to-memory port incorporates:

- 2 DMA channels (read and write)
- Internal to internal transfers
- Data engine for DTCP applications (only for special part numbers)

Functional Description

The memory-to-memory DMA controller is capable of transferring 64-bit bursts of data between internal memories.

The MTM controller supports data in normal word address space only (32-bit). External DMA transfers are not supported.

DMA Channels

Two DMA channels are used for memory-to-memory DMA transfers. The write DMA channel has higher priority over the read channel. The transfer is started by a write DMA to fill up the MTM buffer with a 2 x 32-bit word. Next, the buffer is read back over the same IOD bus to the new destination. With a two position deep buffer and alternate write and read access over the same bus, throughput is limited. The memory-to-memory DMA control register (`MTMCTL`) allows programs to transfer blocks of

64-bit data from one internal memory location to another. This register also allows verification of current DMA status during writes and reads.

Buffer

The MTMFLUSH bit in the MTMCTL register can be set to flush the FIFO and reset the read/write pointers. Setting and resetting the MTMDEN bit only starts and stops the DMA transfer, so it is always better to flush the FIFO along with MTMDEN reset.

Note that the MTMFLUSH bit should not be set along with the MTMDEN bit set. Otherwise the FIFO is continuously flushed leading to DMA data corruption.

Interrupts

There are two DMA channels; one write channel and one read channel. However since both DMA channels are not data independent, only one interrupt is triggered at the DMA transfer end (P15I) if the MTMI bit in the IMASK register is enabled.

Data Throughput

Data throughput for internal to internal transfers is 12 PCLK cycles for 64-bit data.

Programming Model

This data transfer can be set up using the following procedure.

1. Program the DMA registers for both channels.
2. Set (=1) the MTMFLUSH bit (bit 1) in the MTMCTL register to flush the FIFO and reset the read/write pointers.
3. Set (=1) the MTMEN bit in the MTMCTL register.

   A two-deep, 32-bit FIFO regulates the data transfer through the DMA channels

Programming Example

Listing 3-1. Memory-to-Memory DMA

```
#include <def21364.h>
#include <sru21364.h>
#include <SRU.h>

/* Buffer Declarations */
.section/dm seg_dmda;
.align 2;
.var dest[100];
.align 2;
.var source[100];
/* Main code section */
.global _main;
.section/pm seg_pmco;
_main:
r0=0x11111111;
i0=source;
```

```
/* Fill the source buffer */
lcntr=LENGTH(source), do fill until lce;
dm(i0,1)=r0;
fill: r0=rot r0 by 1;

/* Set the interrupt mask for MTMDMA */
bit set imask MTMI;
bit set mode1 IRPTEN;
/* Flush the MTMDMA FIFO */
r0=MTMFLUSH;
dm(MTMCTL)=r0;
/* Set up the source address to read */
r0=source;
dm(IIMTMR)=r0;
/* Set up the destination address to write */
r0=dest;
dm(IIMTMW)=r0;
/* Read and write sequentially with a step of 1 */
r0=1;
dm(IMMTMW)=r0;
dm(IMMTMR)=r0;
/* Read the number of words in source */
r0=length(source);
dm(CMTMR)=r0;
/* Write the number of words in destination */
r0=length(dest);
dm(CMTMW)=r0;
/* Enable MTMDMA */
r0=MTMEN;
dm(MTMCTL)=r0;
_main.end: jump(pc,0);
```

4 PARALLEL PORT

The ADSP-2136x processor has a parallel port that allows bidirectional transfers between it and external parallel devices. Using the parallel port bus and control lines, the processor can interface to 8-bit or 16-bit wide external memory devices. The parallel port provides a DMA interface between internal and external memory and has the ability to support core driven data transfer modes (see Table 4-1).

Table 4-1. Parallel Port Feature Summary

Feature	Availability
Connectivity	
Multiplexed Pinout	Yes
SRU DAI Required	No
SRU DAI Default Routing	N/A
Interrupt Default Routing	Yes (P9I)
Protocol	
Master Capable	Yes
Slave Capable	No
Transmission Simplex	Yes
Transmission Half Duplex	Yes
Transmission Full Duplex	No
Access Type	
Data Buffer	Yes
Core Data Access	Yes

Table 4-1. Parallel Port Feature Summary (Cont'd)

Feature	Availability
DMA Data Access	Yes
DMA Channels	1
DMA Chaining	Yes
Interrupt Source	Core/DMA
Boot Capable	No
Local Memory	No
Clock Operation	PCLK/3

Features

- Support for standard SRAMs
- Interface requires only 16 Pins for address and data
- Total pin count of 19
- Programmable wait and hold cycles

The processor provides two data packing modes for the parallel port, 8/32 and 16/32. For reads, data packing invdves shifting multiple successive 8- or 16-bit elements from the parallel port to the ADSP-2136x processor's receive register to form each 32-bit word, transferring multiple successive 8-bit or 16-bit elements. For writes, packing involves shifting each 32-bit word out into 8- or 16-bit elements that are placed into the memory device.

The parallel port on the ADSP-2136x processors may only move 32-bit data to and from internal memory (normal word addressing).

This chapter describes, in order, the hardware (pins), the basic function of the peripheral, registers, basic operation including DMA and core data

transfers, and the programming model, with a programming example. Figure 4-1 shows a block diagram of the parallel port.

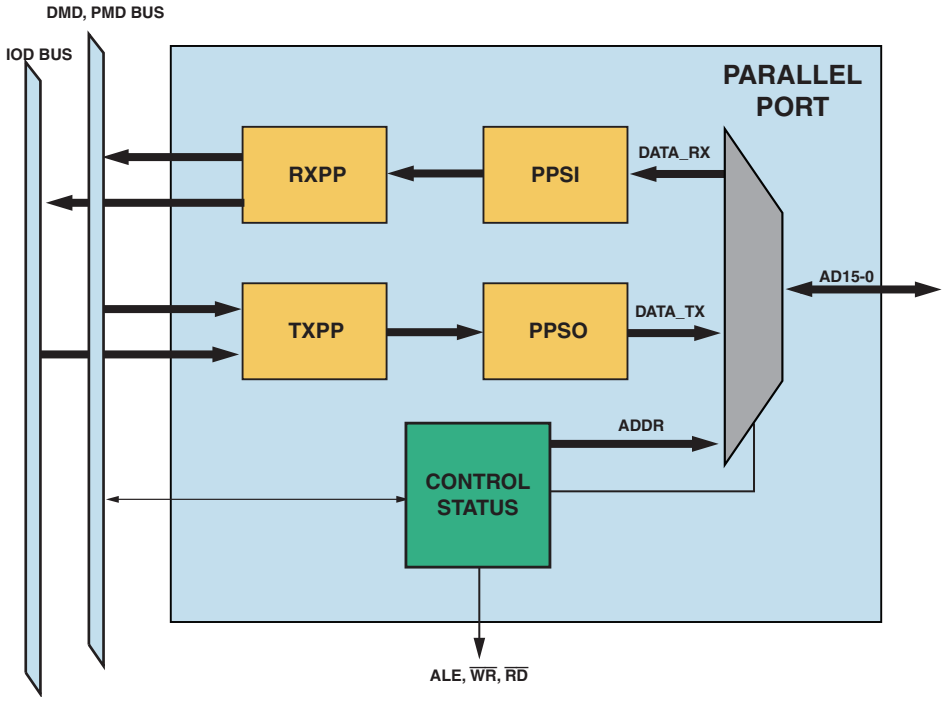


Figure 4-1. Parallel Port Block Diagram

Pin Descriptions

For a complete list of pin descriptions and package pinouts, see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

Unlike previous SHARC processors with an external port, the ADSP-2136x processors do not have a memory select (MS) pin. The boot prom's chip select (CS) should be generated from an address decoder, or otherwise derived from the parallel port signals.

Multiplexed Pin Functions

The parallel port pins can function as flag pins and the parallel data acquisition port pins can function as address pins. For complete information on the pin multiplexing scheme used with these processors, see "Parallel Port Pin Multiplexing" on page 14-20.

Functional Description

This section describes how the parallel port transfers data. The SYSCTL and PPCTL registers control the parallel port operating mode. The bits in the SYSCTL register are listed in Table A-1 on page A-4. Table A-3 on page A-12 lists all the bits in the PPCTL register.

Multiplexed Operation

A parallel port multiplexed external transaction consists of a combination of an address cycle (ALE cycle) and a data cycle (a read or write cycle). This section describes the parallel port operation as it relates to processor timing. Refer to the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet* for detailed timing specifications.

Address Cycles

An ALE cycle is an address latch cycle. It is activated one cycle prior to the address. In this cycle the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals are inactive and ALE is strobed. The upper 16 bits of the address are driven onto the AD15-0 lines for two PCLK cycles, and shortly thereafter the ALE pin is strobed, with AD15-0 remaining valid slightly after deassertion to ensure a sufficient hold time for the external latch. The ALE pin always remains high for 2 x PCLK, (peripheral clock cycles) irrespective of the data cycle duration values that are set in the PPCTL register. The parallel port runs at 1/3 the PCLK rate, and so the ALE cycle is 3 x PCLK. An ALE cycle is inserted whenever the upper 16 bits of address differs from a previous access, as well as after the parallel port is enabled.

The ALE pin is active high by default, but can be set active low via the PPALEPL bit (bit 13) in the parallel port control (PPCTL) register.

Since ALE polarity is active high by default, systems using parallel port boot mode must use address latching hardware that can process this active high signal. For complete information on using the parallel port for booting, see "Parallel Port Booting" on page 14-36.

Data Cycles

In a read cycle, the $\overline{\text{WR}}$ and ALE signals are inactive and the $\overline{\text{RD}}$ signal is strobed. If the upper 16 bits of the external address have changed, this cycle is always preceded by an ALE cycle. In 8-bit mode, the lower 8 bits of the address, A7-0, are driven on the AD15-8 pins, and data is latched from the AD7-0 pins with the rising edge of $\overline{\text{RD}}$. In 16-bit mode, address bits are not driven in the read cycle, the external address is provided entirely by the external latch, and data is latched from the AD15-0 pins with the rising edge of $\overline{\text{RD}}$. Read cycles can be lengthened by configuring the parallel port data cycle duration bits in the PPCTL register.

In a write cycle, the $\overline{\text{RD}}$ and ALE signals are inactive and the $\overline{\text{WR}}$ signal is strobed. If the upper 16 bits of the external address have changed, this cycle is always preceded by an ALE cycle. In 8-bit mode, the lower 8 bits of the address are driven on the AD15-8 pins and data is driven on the AD7-0 pins. In 16-bit mode, address bits are not driven in the write cycle. The external address is provided entirely by the external latch, 16-bit data is driven onto the AD15-0 pins, and data is written to the external device with the rising edge of the $\overline{\text{WR}}$ signal. Address and data are driven before the falling edge of $\overline{\text{WR}}$ and deasserted after the rising edge to ensure enough setup and hold time with respect to the $\overline{\text{WR}}$ signal. Write cycles can be lengthened by configuring the parallel port data cycle duration bits in the PPCTL register.

Data Buffers

The parallel port has two data buffers or FIFOs, one each for reads and writes. These are explained the following sections.

Read Path

The parallel port has a two stage data FIFO for receiving data (RXPP). In the first stage, a 32-bit register (PPSI) provides an interface to the external data pins and packs the 8- or 16-bit input data into 32 bits. Once the 32-bit data is received in PPSI, the data is transferred into the second 32-bit register (RXPP). Once the receive FIFO is full, the chip cannot initiate any more external data transfers. The RXPP register acts as the interface to the core or I/O processor (for DMA).

Note that the PPTRAN bit must be zero in order to perform an external read.

The order of 8- to 32-bit data packing is shown in Table 4-2. The first byte received is 7-0, the second 15-8, and so on. The 16- to 32-bit packing scheme is shown in the third column of the table.

Table 4-2 does not show ALE cycles; it shows only the order of the data reads and writes.

Table 4-2. Packing Sequence for 32-bit Data

Transfer	AD7–0, 8-bit to 32-bit (8-bit bus, LSW first)	AD15–0, 16-bit to 32-bit (16-bit bus, LSW first)
First	Word 1; bits 7–0	Word 1; bits 15–0
Second	Word 1; bits 15–8	Word 1; bits 31–16
Third	Word 1; bits 23–16	
Fourth	Word 1; bits 31–24	

Write Path

The parallel port has a two stage data FIFO for transmitting data (TXPP). The first stage (TXPP) is a 32-bit register that receives data from the internal memory via the DMA controller or a core write. The data in TXPP is moved to the second 32-bit register, PPSO. The PPSO register provides an interface to the external pins. Once a full word is transferred out of PPSO, TXPP data is moved to PPSO, if TXPP is not empty.

Note that the PPTRAN bit in the PPCTL register must be set to one in order to enable external writes.

The order of 32- to 8-bit data unpacking is shown in Table 4-3. The first byte transferred from PPSO is 7-0, the second 15-8, and so on. The 32-bit to 16-bit unpacking scheme is shown in column three of the table.

Table 4-3 does not show ALE cycles; it shows only the order of the data reads and writes.

Table 4-3. Unpacking Sequence for 32-bit Data

Transfer	AD7–0, 32-bit to 8-bit (8-bit bus, LSW first)	AD15–0, 32-bit to 16-bit (16-bit bus, LSW first)
First	Word 1; bits 7–0	Word 1; bits 15–0
Second	Word 1; bits 15–8	Word 1; bits 31–16
Third	Word 1; bits 23–16	
Fourth	Word 1; bits 31–24	

Parallel port DMAs may only be performed to 32-bit (normal word) internal memory.

Operation Modes

The external interface follows the standard asynchronous SRAM access protocol. The programmable data cycle duration bit (PPDUR) and optional bus hold cycle bit (BHC) are provided to interface with memories having different access time requirements. The data cycle duration is programmed via the PPDUR bit in the PPCTL register. The hold cycle at the end of the data cycle is programmed via the PPBHC bit in the PPCTL register.

Disabling the parallel port (PPEN bit is cleared) flushes both parallel port FIFOs, RXPP, and TXPP.

For standard asynchronous SRAM there are two transfer modes—8-bit and 16-bit mode. In 8-bit mode, the address range is 0x0 to 0xFFFFFF which is 16M bytes (4M 32-bit words). In 16-bit mode, the address range is 0x0 to 0xFFFF which is a 128K bytes (32K 32-bit words). Although programs can initiate reads or writes on one and two byte boundaries, the parallel port always transfers 4 bytes (two 16-bit or four 8-bit words).

8-Bit Mode

An ALE cycle always precedes the first transfer of data after the parallel port is enabled. During ALE cycles for 8-bit mode, the upper 16 bits of the external address (A23-8) are driven on the 16-bit parallel port bus (pins AD15-0). In data cycles (reads and writes), the processor drives the lower 8 bits of address A7-0 on the AD15-8 pins. The 8 bits of external data, D7-0, that are provided by AD7-0 are sampled/driven on the rising edge of the $\overline{RD}/\overline{WR}$ signal. The processor continues to receive and or send data with the same ALE cycle until the upper 16 bits of external address differ from the previous access. For consecutive accesses (parallel port DMA external address modifier register, EMPP = 1), this address change occurs once every 256 cycles. Figure 4-3 shows the connection diagram for the 8-bit mode.

In 8-bit mode, a maximum of 24 bits of external address are facilitated through latching the upper 16 bits, A23-8, from AD15-0 into the external latch during the ALE phase of the cycle. The remaining 8 bits of address

A7-0 are provided through AD15-8 during the second half of the cycle when the $\overline{RD}$ or $\overline{WR}$ signal is asserted. The AD7-0 bits provides the data during the same cycle when $\overline{RD}$ or $\overline{WR}$ is asserted.

Eight-bit mode enables a larger external address range.

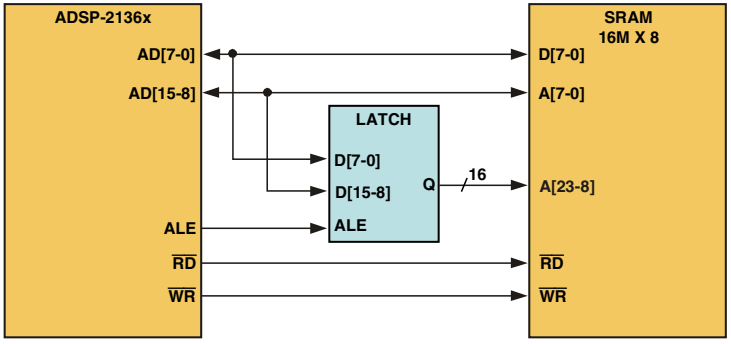


Figure 4-2. External Transfer–8-bit Mode

16-Bit Mode

In 16-bit mode, the external address range is A15-0 (64K addressable 16-bit words). For a nonzero stride value (EMPP ≠ 0), the transfer of data occurs in two cycles. In cycle one, the processor performs an ALE cycle, driving the 16 bits of external address, A15-0, onto the 16-bit parallel port bus (pins AD15-0), allowing the external latch to hold this address. In the second cycle, the processor either drives or receives (on a read/write cycle) the 16 bits of external data (D15-0) through the 16-bit parallel port bus (pins AD15-0). This pattern repeats until the transfer completes.

However, a special case occurs when the external address modifier is zero, (EMPP = 0). In this case, the external address is latched only once, using the ALE cycle before the first data transfer. After the address has been latched externally, the processor continues receiving and sending 16-bit data on AD15-0 until the transfer completes. This mode can be used with external

FIFOs and high speed A/D and D/A converters and offers the maximum throughput available on the parallel port (111M byte/sec).

In 16-bit mode, 16 bits (maximum) of external address are available through latching the 16 bits of A15-0 from AD15-0 into the external latch during the ALE phase of the cycle. The AD15-0 bits represent the external 16 bits of data during the second half of the cycle when the $\overline{RD}$ or $\overline{WR}$ signal is asserted.

The ALE signal is deasserted one peripheral clock cycle (PCLK) after the address is driven and one peripheral clock cycle before the address is received. This provides enough setup and hold time for the 16-bit address with respect to ALE.

Figure 4-3 shows the connection diagram in 16-bit mode.

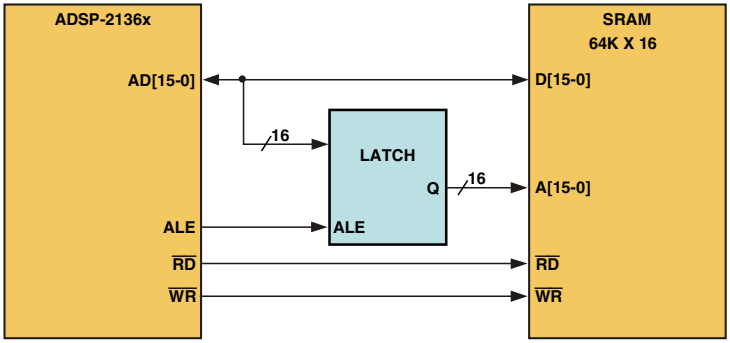


Figure 4-3. External Transfer–16-bit Mode

Parallel Port Registers

The ADSP-2136x processor's parallel port contains several user-accessible registers. The parallel port control register, PPCTL, contains control and status bits and is described below.

Control Register (PPCTL)

The parallel port control (PPCTL) register is a memory-mapped register and is used to configure and enable the parallel port system. This register also contains status information for the Tx/Rx FIFO, the state of DMA, and for external bus availability. This read/write register is also used to program the data cycle duration and to determine the data transfer format.

Table A-3 on page A-12 provides the bit descriptions for the PPCTL register.

Data Buffer Register (RXPP/TXPP)

Two additional registers, RXPP and TXPP, are used for buffering receive and transmit data during DMA operations and can be accessed by the core.

Data Transfer Types

There are a number of considerations to make when interfacing to parallel external devices. This section describes the different ways that the parallel port can be used to access external devices. Considerations for choosing between an 8-bit or a 16-bit wide interface are discussed in "8-Bit Versus 16-Bit SRAM Modes" on page 4-22.

External parallel devices can be accessed in two ways, either using DMA-driven transfers or core-driven transfers. DMA transfers are performed in the background by the I/O processor and are generally used to move blocks of data. To perform DMA transfers, the address, word-count, and address modifier are specified for both the source and destination buffers (one internal, one external). Once initiated, (by setting PPEN=1 and PPDEN=1), the IOP performs the specified transfer in the background without further core interaction. This is the main advantage of DMA transfers over core-driven transfers is they allow the core to continue executing code while sequential data is imported/exported in the background.

Unlike the external port on previous SHARC processors, the ADSP-2136x core cannot directly access the external parallel bus. Instead, the core initializes two registers to indicate the external address and address modifier and then accesses data through intermediate registers.

DMA Transfers

To use the parallel port for DMA programs, start by setting up values in the DMA parameter registers. The program then writes to the PPCTL register to enable PPDEN with all of the necessary settings like cycle duration value, transfer direction, and so on. While a parallel port DMA is active, the DMA parameter registers are not writeable. Furthermore, only the PPEN and DMAEN bits (in the PPCTL register) can be changed. If any other bit is changed, the parallel port malfunctions. It is recommended that both the PPDEN and PPEN bits be set and reset together to ensure proper DMA operation.

DMA Internal Word Count Register (ICPP)

This 16-bit register contains the number of words in internal memory to be transferred via DMA. There should be a correlation between the ECPP and ICPP values. In 16-bit mode, the ECPP value should be double that of ICPP and for 8-bit mode the ECPP value should be four times that of ICPP. Also, a DMA chain pointer descriptor where ICPP = 0 and/or ECPP = 0 is not allowed. Both of these cases cause the DMA engine to hang.

External Word Count Register (ECPP)

This 24-bit register contains the number of words in external memory to be transferred via DMA. There should be a correlation between the ECPP and ICPP values. In 16-bit mode the ECPP value should be double that of ICPP and for 8-bit mode the ECPP value should be four times that of ICPP. Also, a DMA chain pointer descriptor where ICPP = 0 and/or ECPP = 0 is not allowed. Both of these cases cause the DMA engine to hang.

Before initializing a chain pointer DMA, it is important that ECPP and ICPP are set to zero.

Chained DMA Transfers

DMA chaining is enabled by setting the PPCHEN (bit 30 in PPCTL register). When chaining is enabled, the next set of DMA parameters are loaded from internal memory after the current DMA cycle and new DMA starts. The index of the start of the memory block is stored in the CPPP register. DMA parameter values reside in consecutive memory locations as explained in Table 2-14 on page 2-30. Chaining ends when the CPPP register contains address 0x00000 for the next parameter block.

DMA Chain Pointer Register (PPCP)

This 20-bit register contains the internal memory index for the next transfer control block containing the set of DMA parameters (bits 18–0) and the PCI bit (bit 19). When PCI bit is set, interrupts are generated whenever the current DMA ends. When this bit is not set, an interrupt is generated only at the end of a DMA chain. The DMA chain ends when PPCP18-0 are all zeros.

For more information, see “TCB Memory Storage” on page 2-20.

Unlike the ADSP-2126x SHARC processors, the ADSP-2136x does support DMA chaining for parallel port. Before initializing a chain pointer DMA, it is important that ECPP and ICPP are set to zero.

DMA Transfer Rules

The following are rules that should be followed when using the parallel port to perform DMA transfers. Note that in all cases, failure to adhere to these restrictions causes the DMA engine to hang, a loss of data, or other unpredictable behavior.

- A DMA transfer can be interrupted by resetting the PPDEN bit, but none of the other control settings (except for the PPEN bit) should be changed. If the parallel port remains enabled, then interrupted DMA can be resumed by setting the PPDEN bit again.
- Resetting the parallel port during a DMA operation is prohibited. If the parallel port is disabled by resetting the PPEN bit, data in FIFO is flushed.
- Before initializing DMA chaining, it is important that the ECPP and ICPP registers are zero.
- There should be a correlation between the values in the ECPP and ICPP registers. For example, in 16-bit mode, the value in the ECPP register should be double that of the value in the ICPP register. In 8-bit mode, the value in the ECPP register should be four times that of the value in the ICPP register.
- A DMA descriptor where ICPP = 0 and/or ECPP = 0 is not allowed and causes the DMA engine to hang.
- Do not disable chaining when a chained DMA transfer is in progress (the PPCHEN bit in the PPCTL register = 0). If attempted, then programmers should be aware of the consequences and should not expect a DMA completion interrupt in case of PCI = 0.

Core-Driven Transfers

The following registers must be initialized for core-driven transfers

DMA external index address register (EIPP). This 24-bit register contains the external memory byte address used for core-driven transfers.

DMA external address modifier register (EMPP). This 2-bit register contains the external memory address modifier. It supports only +1, 0, –1. After each data cycle, the EIPP register is modified by this value.

When the core accesses either the TXPP or RXPP registers, the parallel port writes/fetches data to/from the specified external address. The details of this functionality and the four main techniques to manage each transfer are detailed in the following sections. In general, core-driven transfers are most advantageous when performing single-word accesses and/or accesses to non-sequential addresses.

Non sequential core transfers require the following procedure.

1. Disable the parallel port
2. Check the interface status to ensure the port is idle
3. Write new values to both parameter registers (EIPP and ECPP)
4. Re-enable the parallel port

Core-driven transfers can be managed using four transfer techniques.

1. Interrupt driven
2. Status driven
3. Known duration
4. Core stall

For all four of these methods, the core uses the same basic steps to initiate the transfer. However, each method uses a different technique to complete the transfer.

Interrupt Driven Accesses

With interrupt-driven accesses, parallel port interrupts are generated on a word-by-word basis, rather than on a block transfer basis, as is the case with DMA. In this non-DMA mode, the interrupt indicates to the core that it is now safe to read a word from the RXPP buffer or to write a word to the TXPP buffer (depending on the value of the PPTRAN bit).

To facilitate this, the PPI (latch) bit of the LIRPTL register is set to one in every core cycle where the TXPP buffer is not full or, in receive mode, in every core cycle in which the RXPP buffer has valid data. When fast 16-bit wide parallel devices are accessed, there may be as few as ten core cycles between each transfer. Because of this, interrupt-driven transfers are usually the least efficient method to use for core-driven accesses. Interrupt-driven transfers are most valuable when parallel port data cycle durations are very long (allowing the core to do some work between accesses). Generally, interrupts are the best choice for DMA-driven parallel port transfers rather than core-driven transfers.

Status-Driven Transfers (Polling)

The second method that the core mayuse to manage parallel port transfers involves the status bits in PPCTL register, specifically the parallel port bus status (PPBS) bit. This bit reflects the status of the external address pins AD0-AD15 and is used to determine when it is safe to disable and modify the parallel port. The PPBS bit is set to 1 at the start of each transfer and is cleared once the entire 32-bit word has been transmitted/received.

Known-Duration Accesses

Of the four core-driven data transfer methods, known duration accesses are the most efficient because they allow the core to execute code while the transfer to/from the RXPP or TXPP occurs on the external bus. For example, after the core reads the PPTX register, it takes some number N core cycles for the parallel port to shift out that data to the memory. During that time, the core can go on doing other tasks. After N core cycles have passed, the parallel port may be disabled and the external address register updated for another access.

To determine the duration for each access, the designer simply adds the number of data cycles and the duration of each (measured in CCLK cycles) along with the number of ALE cycles (which are fixed at three PCLK cycles). This duration is deterministic and based on two settings in the PPCTL

register—parallel port data cycle duration (PPDUR) and bus hold cycle enable (PPBHC).

Please refer to "Functional Description" on page 4-4 for further explanation of the parallel port bus cycles, but in summary, programs can use the following values.

- Each ALE cycle is fixed at three PCLK cycles, regardless of the PPDUR or PPBHC settings.
- Each data cycle is the setting in the PPDUR register (+1 if PPBHC = 1)

For example, in 8-bit mode, a singleword transfer is comprised of one ALE cycle and four data cycles. If PPDUR3 is used (the fastest case) and PPBHC = 0, this transfer completes in:
(1 ALE cycle x 3 PCLK) + (4 data cycles x 3 PCLK) = 15 PCLK cycles = 30 CCLK cycles per 32-bit word. This means that 30 instructions after data is written to TXPP or read from RXPP, the parallel port has finished writing/fetching that data externally, and the parallel port may be disabled. This case is shown in Listing 4-1 on page 4-26.

Core-Stall Driven Transfers

The final method of managing parallel port transfers simply relies on the fact that the core stalls execution when reading from an empty receive buffer and when writing to a full transmit buffer. This technique can only be used for accesses to sequential addresses in external memory. For sequential external addresses, the parallel port does not need to be disabled after each word in order to manually update the EIPP register. Instead, the external address that is automatically incremented by the modifier (EMPP) register on each access is used.

The following are guidelines that programs must follow when the processor core accesses parallel port registers.

- While a DMA transfer is active, the core may only write the PPEN and PPDEN bits of PPCTL. Accessing any of the DMA parameter registers or other bits in PPCTL during an active transfer will cause the parallel port to malfunction.
- Core reads of the FIFO register during a DMA operation are allowed but do not affect the status of the FIFO.

  If PPEN is cleared while a transfer is underway (whether core or DMA driven), the current external bus cycle (ALE cycle or data cycle) completes but no further external bus cycles occur. Disabling the parallel port clears the data in the RXPP and TXPP registers.
- Core reads and writes to the TXPP and RXPP registers update the status of the FIFO when DMA is not active. This happens even when the parallel port is disabled.
- For core-driven transfers over the parallel port, the IIPP, IMPP, ICPP, and ECPP registers are not used. Only the EIPP and EMPP registers need to be initialized before accessing the TXPP or RXPP buffers.
- To change any access related control bits in the PPCTL register, first disable the parallel port by clearing the PPEN bit in the PPCTL register, then read the PPBS bit in the PPCTL register to check if the external interface of parallel port is an idle state, and finally write to the PPCTL register with the new control settings.

Interrupts

The parallel port has one interrupt signal (PPI) which is generated for all core or DMA related operations.

DMA Interrupts

When DMA is enabled, the maskable interrupt PPI occurs when the DMA block transfer has completed (when theDMA internal word count register ICPP decrements to zero).

When DMA chaining is enabled and the PCI bit is set in the CPPP register, interrupts are generated whenever the current DMA ends. If this bit is not set, then the interrupt is generated only at the end of the DMA chain. The DMA chain ends when CPPP18-0 are all zeros.

Core Interrupts

When DMA is disabled, the maskable interrupt is latched in every cycle the receive buffer is not empty or the transmit buffer is not full.

Throughput

As described in "Functional Description", each 32-bit word transferred through the parallel port takes a specific period of time to complete. This throughput depends on a number of factors, namely parallel port speed, memory width (8 bits or 16 bits), and memory access constraints (occurrence of ALE cycles at page boundaries, duration of data cycles, and/or addition of hold time cycles).

The maximum parallel port speed is one-third (1/3) of the peripheral clock. The relationship between core clock and parallel port speed is static. For a 333 MHz core clock, the peripheral clock is 166.5 MHz, and the parallel port runs at 55 MHz. Since there is no parallel port clock signal, it is easiest to think of parallel port throughput in terms of peripheral clock cycles:

(333 MHz/2)/3 = 55.5 MBytes/sec

As described in "Functional Description" on page 4-4, parallel port accesses require both ALE cycles to latch the external address and additional data cycles to transmit or receive data. Therefore, the throughput on the parallel port is determined by the duration and number of these cycles per word. The duration of each type of cycle is shown below and the frequency is determined by the external memory width.

There is one case where the frequency is also determined by the external address modifier register (EMPP).

- All ALE cycles are fixed at three peripheral clock cycles (PCLK) and are not affected by the PPDUR or BHC bit settings. In this case, the ALE signal is high for two peripheral clock cycles. The address for the ALE is set up one-half (1/2) peripheral clock cycle before ALE goes HIGH (active) and remains on the bus one-half (1/2) cycle after ALE goes LOW (inactive). Therefore, the total ALE cycles on the bus are 1/2 + 2 + 1/2 = 3 PCLK cycles. Please refer to the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet* for more precise timing characteristics.
- Data cycle duration is programmable with a range of 3 to 32 PCLK cycles. They may range from 4 to 33 cycles if the BHC bit is set (=1)

The following sections show examples of transfers that demonstrate the expected throughput for a given set of parameters. Each word transfer sequence is made up of a number of data cycles and potentially one additional ALE cycle.

8-Bit Access

In 8-bit mode, the first data-access (whether a read or a write) always consists of one ALE cycle followed by four data cycles. As long as the upper 16 bits of address do not change, each subsequent transfer consists of four data cycles. The ALE cycle is inserted only when the parallel port address crosses an 8-bit boundary page, in other words, after every 256 bytes that are transferred.

For example, assume PPDUR3, BHC = 0, and the parallel port is in 8-bit mode. The first byte on a new page takes six peripheral clock cycles (three for the ALE cycle and three for the data cycle), and the next sequential 255 bytes consume three peripheral clock cycles each. Therefore, the average data rate for a 256 byte page is:

(3 PCLK x 255 + 6 PCLK x 1)/256 = 3 PLCK/byte

For a 333 MHz core, this results in:

(166 MHz PCLK) x (1 byte/3 PCLK) = 55.3M Bytes/sec

There should be a correlation between the ECPP and ICPP register values. In 8-bit mode, the ECPP value should be four times that of ICPP.

16-Bit Access

In 16-bit mode, every word transfer consists of two ALE cycles and two data cycles. Therefore, for every 32-bit word transferred, at least six PCLK cycles are needed to transfer the data plus an additional six PCLK cycles for the two ALE cycles, for a total of 12 PCLK cycles per 32-bit transfer (four bytes). For a 333 MHz core clock, this results in a maximum sustained data rate device of:

166 MHz/3 = 55.5M Bytes/sec

There is a specific case which allows this maximum rate to be exceeded. If the external address modifier (EMPP) is set to a stride of zero, then only one ALE cycle is needed at the very start of the transfer. Subsequent words, essentially written to the same address, do not require any ALE cycles, and every parallel port cycle may be a 16-bit data cycle. In this case, the throughput is nearly doubled (except for the very first ALE cycle) to over 111M bytes per second. This mode is particularly useful for interfacing to FPGA's or other memory-mapped peripherals such as DAC/ADC converters.

There should be a correlation between the ECPP and ICPP register values. In 16-bit mode, the ECPP value should be double that of ICPP.

8-Bit Versus 16-Bit SRAM Modes

When considering whether to employ the 16- or 8-bit mode in a particular design, a few key points should be considered.

- The 8-bit mode provides a 24-bit address, and therefore can access 16M bytes of external memory. By contrast, the 16-bit mode can only address 64K x 16 bit words, which is equivalent to 128K bytes. Therefore, the 8-bit mode provides 128 times more storage capacity then the 16-bit mode.
- For sequential accesses, the 8-bit mode requires only one ALE cycle per 256 bytes. With minimum wait states selected, this represents a worst case overhead of:
  (1 ALE cycle)/(256 accesses + 1 ALE) x 100% = 0.39% overhead for ALE cycles. In contrast, the 16-bit mode requires one ALE cycle per external sequential access. Regardless of length (N), this represents a worst case overhead of:
  (N ALE cycles)/(N accesses + N ALE cycles) x 100% = 50% overhead for ALE cycles. However, the 16-bit mode delivers two bytes per cycle. Therefore, the total data transfer speed for sequential accesses is nearly identical for both 8-bit and 16-bit modes.

It seems that since the total transfer rates are the same for both 8-bit and 16-bit modes, and the 8-bit mode can also address 128 times as much external memory, 8-bit mode would always be preferred. However, the following should be considered.

- Some external devices are only capable of interfacing to a 16-bit bus.

- When the DMA external modifier is set to zero, (EMPP = 0), the address does not change after the first cycle, therefore an ALE cycle is only inserted on the first cycle. In this case, the 16-bit port can run twice as fast as the 8-bit port, as the overhead for ALE cycles is zero. This is convenient when interfacing to high speed 16-bit FIFO-based devices, including A/D and D/A converters.
- In situations where a majority of address accesses are non-sequential and cross 256 byte boundaries, the overhead of the ALE cycles in the 8-bit mode approaches 20%[1]. In this particular situation, the 16-bit memory can provide a 40% speed advantage over the 8-bit mode.

Parallel Port Effect Latency

The PPCTL register has a two-cycle effect latency. This means that if programs write to this register in cycle N, the new settings are not in effect until cycle N + 2. Avoid sampling PPBS until at least two cycles after the PPEN bit in PPCTL is set.

For read operations (PPTRAN=0), two core clock cycles after PPEN is set (=1), the parallel port fetches two 32-bit data words from the external byte address indicated by EIPP. Subsequently, additional data is fetched only when the core reads (empties) RXPP.

Programming Model

The following sections provide information for setting up and using the parallel port.

[1] This can be realized by recalling that four bytes must be packed/unpacked into a single 32-bit word. For example when a 32-bit word is written/read, there is a single ALE cycle inserted per four consecutive addresses. This results in: (N/4 ALE cycles)/(N accesses + N/4 ALE cycles) x 100% = 20%.

Configuring the Parallel Port for DMA

Use the following steps to configure the parallel port for a standard DMA transfer.

1. Set (or reset) the PPTRAN bit in the PPCTL register. Depending on whether the operation is write or read, ensure that FIFO is empty and the external interface is idle by reading the status of the PPS and PPBS bits respectively.
2. Initialize the IIPP, IMPP, ICPP, EIPP, EMPP and ECPP registers with the appropriate values, keeping PPDEN and PPEN disabled.
3. With all other controls set in the PPCTL register, enable the PPEN and PPDEN bits.

Configuring a Chained DMA

Use the following steps to configure the parallel port for a chained DMA transfer.

1. Before initializing DMA chaining, clear the ECPP and ICPP registers to zero
2. Set (or reset) the PPTRAN bit. Depending on whether the operation is write or read, ensure that the FIFO is empty and that the external interface is idle by reading the status of the PPS and PPBS bits respectively.
3. Initialize the CPPP register with the address of the next transfer control block. Set the PCI bit if the interrupt is needed at the end of each DMA block transfer.
4. Set the PPCTL register with all the required controls and enable the PPEN, PPDEN and PPCHEN bits. Once DMA chaining is enabled, the DMA engine fetches the IIPP, IMPP, ICPP, EIPP, EMPP and ECPP register values from the memory address specified in the CPPP

register. Once the DMA descriptors are fetched, normal DMA execution starts and continues until the CPPP register contains all zeros.

Configuring the Parallel Port for Core Access

The following steps provide the basic procedure for setting up and initiating a data transfer using the core.

1. Before initializing or modifying any of the parallel port parameter registers such as EIPP and EMPP, the parallel port must first be disabled (bit 0, PPEN, of the PPCTL register must be cleared). Only when PPEN=0, may those registers be modified and the port then re-enabled. This sequence is most often used to perform non-sequential, external transfers, such as when accessing taps in a delay line.

2. Write the external byte address to the EIPP register and the external address modifier to the EMPP register. For core-driven transfers, the ECPP, IIPP, IMPP, and ICPP registers are not used. Although these registers are automatically updated by the parallel port (the ECPP register decrements for example), they may be left un initialized without consequence.

3. Initialize the PPCTL register with the appropriate settings. These include the parallel port data cycle duration (PPDUR) and whether the transfer is a receive or transmit operation (PPTRAN). For core-driven transfers, be sure to clear the DMA enable bit, PPDEN. In this same write to PPCTL, the port may also be enabled by setting bit 0, PPEN, to 1.

When enabling the parallel port (setting PPEN=1), the external bus activity varies, depending on the direction of data transfer (receive or transmit). For transmit operations (PPTRAN=1), the parallel port does not perform any external accesses until valid data is written to the TXPP register by the core.

Programming Examples

The program shown in Listing 4-1 performs a chained DMA.

Listing 4-1. Parallel Port Chained DMA

```
#include <def21364.h>
#include <sru21364.h>
#include <SRU.h>
#define N 5

.section/pm seg_rth;
nop; nop; nop; nop;
nop; jump start; rti; rti;

/* PP interrupt service routine at location 0x00090050 */
.section/pm seg_pp;
jump isr; rti; rti; rti;
/* Enable PP interrupt - By default PPI interrupt is mapped to
P9 interrupt */
bit clr lirptl P9I;
bit set mode1 IRPTEN;
bit set lirptl P9IMSK;

/* Register used for comparison with the flag value in the ISR*/
r15 = 0x1;
r0 = 0x0;
dm(PPCTL) = r0;
r0 = 0x0;
dm(ECPP) = r0;
dm(ICPP) = r0;
r0 = tx_tcb + 6;
dm(CPPP) = r0;
```

```
r0 = PPEN | PPDUR20 | PPDEN | PPTRAN | PPCHEN | PPBHC;
dm(PPCTL) = r0;
nop;
```

5 DIGITAL APPLICATION INTERFACE

The digital application interface (DAI) is comprised of a groups of peripherals and its respective signal routing unit (SRU). The inputs and outputs of the peripherals are not directly connected to external pins. Rather, the SRUs connect the peripherals to a set ofpins and to eachother, based on a set of configuration registers. This allows the peripherals to be interconnected to suit a wide variety of systems. It also allows the ADSP-2136x SHARC processors to include an arbitrary number and variety of peripherals while retaining high levels of compatibility without increasing pin count.

Features

The DAI incorporates a set of peripherals and a very flexible routing (connection) system permitting a large combination of signal flows as shown in Figure 5-1 on page 5-4. A set of DAI-specific registers make such design, connectivity, and functionality variations possible. All routing related to peripheral states for the DAI interface is specified using DAI registers. For more information on pin states, refer to "I/O Pin Buffers" on page 5-7.

Table 5-1. Routing Unit Feature Summary

Feature	DAI
Pin Buffers	
Number	20
Input	Yes
Output	Yes

Table 5-1. Routing Unit Feature Summary (Cont'd)

Feature	DAI
Open-drain	Yes
Three-state	Yes
High Impedance	Yes
Programmable Pull-up	Yes
I/O Level Status Register	Yes
Interrupts	
Interrupt Source	Core (DAILI or DAIHI)
Total Channels	32
Miscellaneous I/O channels	10
Peripheral Channels	22
Clock Operation	PCLK/2

The DAI may be used to connect combimations of inputs to combinations of outputs. This function is performed by the SRU via memory-mapped control registers.

This *virtual connectivity* design offers a number of distinct advantages:

- Flexibility
- Increased numbers and kinds of configurations
- Connections can be made via software—no hard wiring is required

Inputs may only be connected to outputs.

Functional Description

Figure 5-1 shows how the DAI pin buffers are connected via the SRU.

The DAI is comprised of four primary blocks:

- Peripherals (A/B/C) associated with the DAI
- A Signal Routing Unit (SRU)
- DAI I/O pin buffers
- Miscellaneous buffers

The peripherals shown in Figure 5-1 can have up to three connections (if master or slave capable); one acts as signal input, one as signal output and the third as an output enable. The SRUs are based on a group of multiplexers which are controlled by registers to establish the desired interconnects.

The miscellaneous buffers have an input and output and are used for group interconnection.

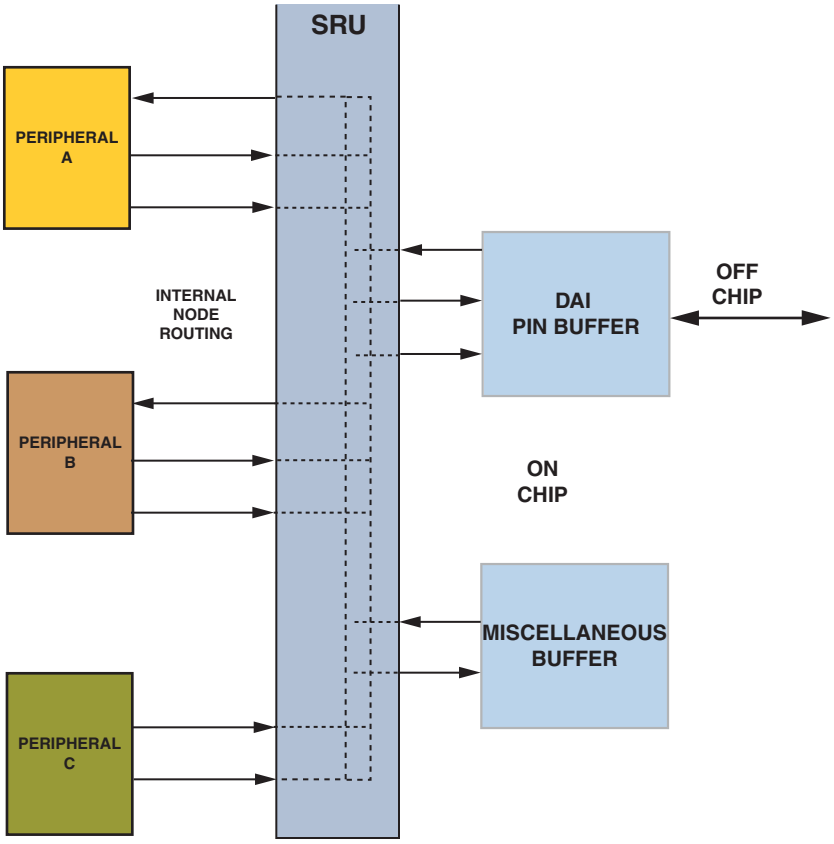


Figure 5-1. DAI Functional Block Diagram

Note that Figure 5-1 is a simplified representation of a DAI system. In a real representation, the SRU and DAI would show several types of data being routed from several sources including the following.

- Serial ports (SPORTs)
- Precision clock generators (PCG)
- Input data port (IDP)

- Asynchronous sample rate converters (SRC)
- S/PDIF transmitter
- S/PDIF receiver
- DAI Interrupts (miscellaneous)

Signal Naming Conventions

Each peripheral associated with the DAI does not have any dedicated I/O pins for off-chip communication. Instead, the I/O pin is only accessible in the chip internally and is known as an *internal node*. Every internal node of a DAI peripheral (input or output) is given a unique mnemonic. The convention is to begin the name with an identifier for the peripheral that the signal is coming to/from followed by the signal's function. A number is included if the DAI contains more than one peripheral type (for example, serial ports), or if the peripheral has more than one signal that performs this function (for example, IDP channels). The mnemonic always ends with _I if the signal is an input, or with _O if the signal is an output (Figure 5-2).

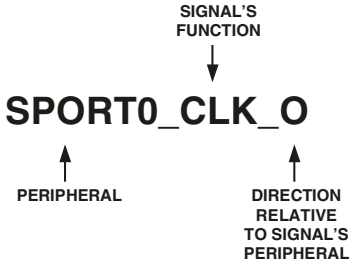


Figure 5-2. Example SRU Mnemonic

DAI Peripherals

All peripherals within the DAI that have bidirectional pins that generate a corresponding pin enable signal. Typically, the settings within a peripheral's control registers determine if a bidirectional pin is an input or an output, and is then driven accordingly. Both the peripheral control registers and the configuration of the SRU can effect the direction of signal flow in a pin buffer.

For example, from an external perspective, when a SPORT is completely routed off-chip, it uses four pins—clock, frame sync, data channel A, and data channel B. Because all four of these pins comprise the interface that the SPORT presents to the SRU, there are a total of 12 connections as shown in Figure 5-3.

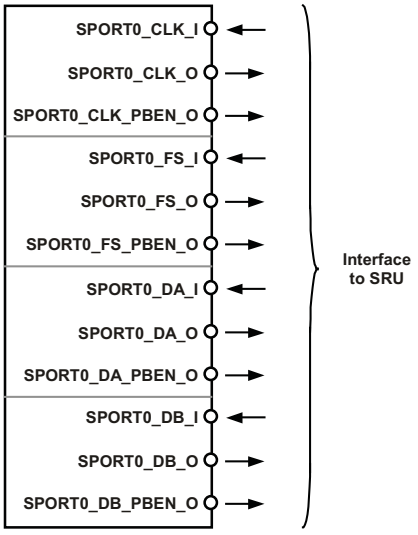


Figure 5-3. SRU Connections for SPORT0

For each bidirectional line, the SPORT provides three separate signals. For example, a SPORT clock has three separate SRU connections (instead of one physical pin):

- input clock to the SPORT (SPORTx_CLK_I)
- output clock of the SPORT (SPORTx_CLK_O)
- output enable clock of the SPORT (SPORTx_CLK_PBEN_O)

For example, if a SPORT's MSTR bit is set in the SPCTLx register, the SPORTx_CLK_O and SPORTx_CLK_PBEN signals are automatically driven. If the MSTR bit is cleared in the SPCTLx register (slave operation), the SPORTx_CLK_O and SPORTx_CLK_PBEN signals are automatically disabled and the SPORTx_CLK_I signal expects an external clock.

Note that the input and output signal pair is never used simultaneously.

The pin enable signal dictates which of the two SPORT lines appear at the DAI pin at any given time. By connecting all three signals through the SRU, the standard SPORT configuration registers behave as documented in Chapter 6, Serial Ports. The SRU then becomes transparent to the peripheral. Figure 5-3 demonstrates SPORT0 properly routed to DAI pins one through four; although it can be equally well routed to any of the 20 DAI pins.

I/O Pin Buffers

Within the context of the SRU, physical connections to the DAI pins are replaced by a logical interface known as a *pin buffer*. This is a three terminal active device capable of sourcing/sinking output current when its driver is enabled, and passing external input signals when disabled. Each pin has a pin input, output, and enable as shown in Figure 5-4. The inputs and the outputs are defined with respect to the pin, similar to a peripheral device. This is consistent with the SRU naming convention.

The notation for pin input and output connections can be quite confusing at first because, in a typical system, a pin is simply a wire that connects to a device. The manner in which the pins are routed within the SRU requires additional nomenclature. The pin interface's input may be thought of as the input to a buffer amplifier that can drive a load on the physical external lead. The pin interface enable is the input signal that enables the output of the buffer by turning it on when its value is logic high, and turning it off when its value is logic low.

When the pin enable is asserted, the pin output is logically equal to pin input, and the pin is driven. When the pin enable is deasserted, the output of the buffer amplifier becomes high impedance. In this situation, an external device may drive a level onto the line, and the pin is used as an input to the ADSP-2136x processor.

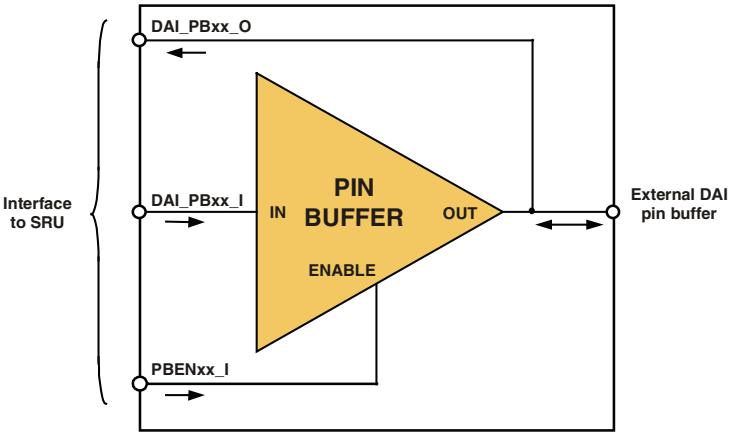


Figure 5-4. Pin Buffer Example

Pin Buffers as Signal Output

In a typical embedded system, most pins are designated as either inputs or outputs when the circuit is designed, even though they may have the ability to be used in either direction. Each of the DAI pins can be used as

either an output or an input. Although the direction of a DAI pin is set simply by writing to a memory-mapped register, most often the pin's direction is dictated by the designated use of that pin. For example, if the DAI pin were to be hard wired to only the input of another interconnected circuit, it would not make sense for the corresponding pin buffer to be configured as an input. Input pins are commonly tied to logic high or logic low to set the input to a fixed value. Similarly, setting the direction of a DAI pin at system startup by tying the pin buffer enable to a fixed value (either logic high or logic low) is often the simplest and cleanest way to configure the SRU.

When the DAI pin is to be used only as an output, connect the corresponding pin buffer enable to logic high as shown in Figure 5-5. This enables the buffer amplifier to operate as a current source and to drive the value present at the pin buffer inputonto the DAI pin and off-chip. When the pin buffer enable (PBENxx_I) is set (= 1), the pin buffer output (PBxx_O) is the same signal as the pin buffer input (PBxx_I), and this signal is driven as an output.

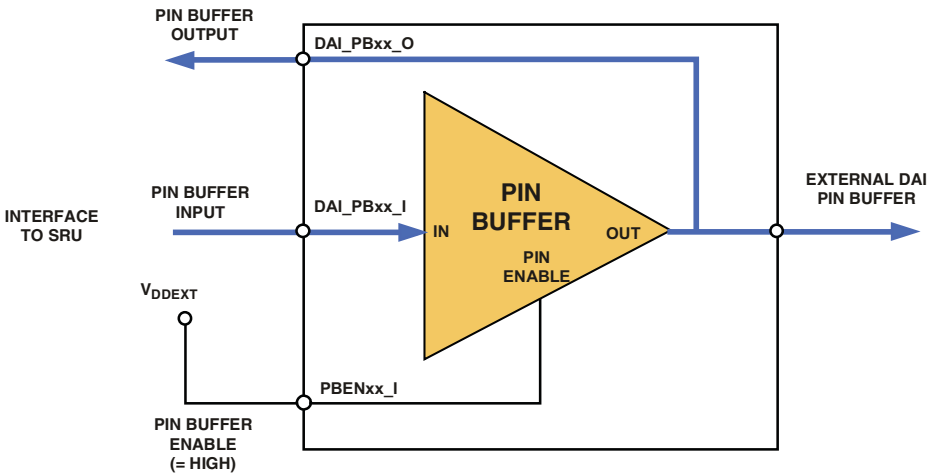


Figure 5-5. Pin Buffer as Output

Pin Buffers as Signal Input

When the DAI pin is to be used only as an input, connect the corresponding pin buffer enable to logic low as shown in Figure 5-6. This disables the buffer amplifier and allows an off-chip source to drive the value present on the DAI pin and at the pin buffer output. When the pin buffer enable (PBENxx_I) is cleared (= 0), the pin buffer output (PBxx_O) is the signal driven onto the DAI pin by an external source, and the pin buffer input (PBxx_I) is not used.

Although not strictly necessary, it is recommended programming practice to tie the pin buffer input to logic low whenever the pin buffer enable is tied to logic low (Figure 5-6 and Figure 5-7).

By default, some pin buffer enables are connected to SPORT pin enable signals that may change value. Tying the pin buffer input low decouples the line from irrelevant signals and can make code simpler to debug. It also ensures that no voltage is driven by the pin if a bug in your code accidentally asserts the pin enable.

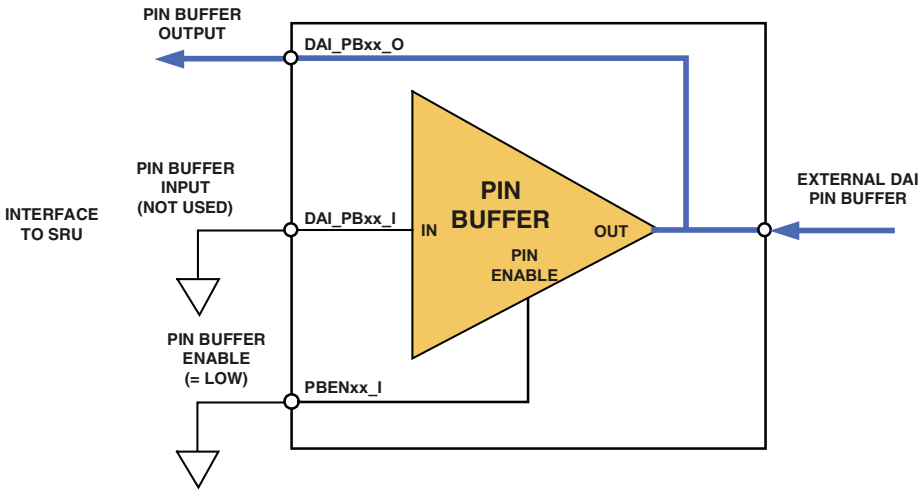


Figure 5-6. Pin Buffer as Input

Programmable Pull-Up Resistors

The pin buffer allows systems to attach a pull-up connected to the pad (high impedance) or disconnected (three state). This is controlled through the DAI_PULLUP register.

Pin Buffers as Open Drain

For peripherals like the SPI (multi processing), the bus protocol requires the pin drivers to work in open drain mode (Figure 5-7) for transmit and receive operation where the signal input of the assigned pin buffer is tied low. The peripheral's data output signal is connected to the PBEN signal. In open drain mode, if PBEN = low, the level on the pin depends on the bus activities. If PBEN = high, the driver is conducting (input always low level) and ties the bus to low level.

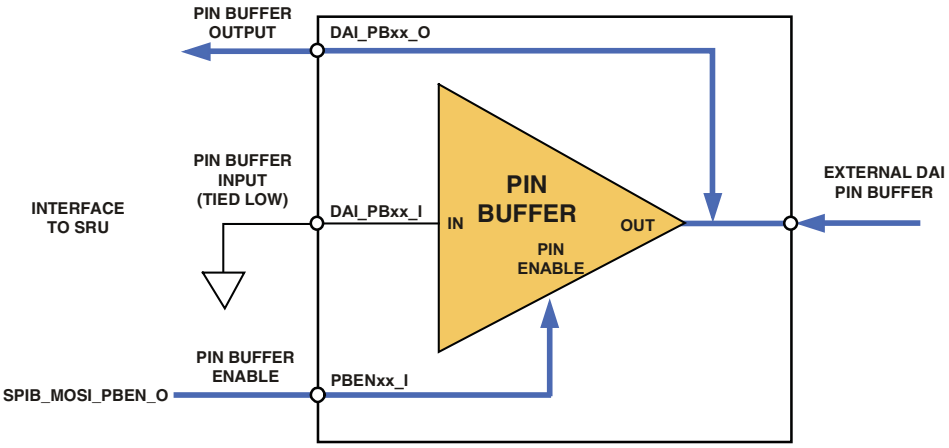


Figure 5-7. Pin Buffer as Open Drain

Miscellaneous Buffers

The miscellaneous buffers are used to interconnect signals from different routing groups. These buffers are similar to the DAI pin buffers with three basic differences.

1. Only for internal connections, no pin buffer enable required
2. MISCxx_O output always feeds DAI interrupt latch register and Group F (PBENx_I)
3. MISCxx_I input sources collected from different groups

The miscellaneous buffers act as intermediate buffer connections between the peripheral's source node and the pin buffer enable destination node. This allows for routing that is not possible among a single group.

The miscellaneous buffer allows interconnects which are not supported within a single DAI routing group.

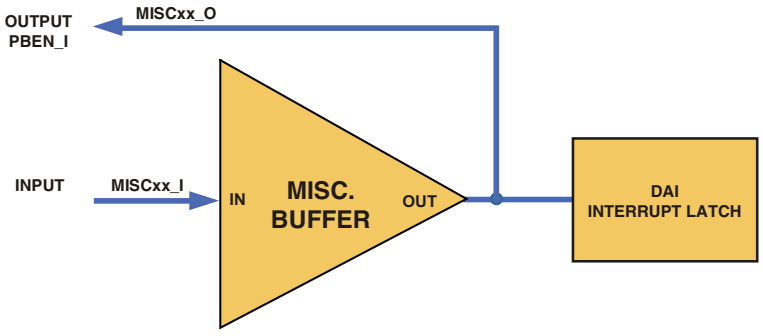


Figure 5-8. Miscellaneous Buffer

Signal Routing Matrix by Groups

The SRU can be compared to a set of patch bays, which contains a bank of inputs and a bank of outputs. For each input, there is a set of permissible output options. Outputs can feed to any number of inputs in parallel, but every input must be patched to exactly one valid output source. Together, the set of inputs and outputs are called a group. The signal's inputs and outputs that comprise each group all serve similar purposes. They are compatible such that almost any output-to-input patch makes functional sense. With the grouping, the multiplexing scheme becomes highly efficient since it wouldn't make sense to route a frame sync signal to a data signal.

The SRU for the DAI contains six groups that are named sequentially A through F. Each group routes a unique set of signals with a specific purpose as shown below.

- Group A routes clock signals
- Group B routes serial data signals
- Group C routes frame sync signals
- Group D routes pin signals
- Group E routes miscellaneous signals
- Group F routes pin output enable signals

Together, the SRU's six groups include all of the inputs and outputs of the DAI peripherals, a number of additional signals from the core, and all of the connections to the DAI pins.

DAI Group Routing

Each group has a unique encoding for its associated output signals and a set of configuration registers. For example, DAI group A is used to route clock signals. The memory-mapped registers, SRU_CLKx, contain bit fields corresponding to the clock inputs of various peripherals. The values written to these bit fields specify a signal source that is an output from another peripheral. All of the possible encodings represent sources that are clock signals (or at least could be clock signals in some systems). Figure 5-9 shows the input signals that are controlled by the group A register, SRU_CLKx. All bit fields in the SRU configuration registers correspond to inputs. The value written to the bit field specifies the signal source. This value is also an output from some other component within the SRU.

The SRU is similar to a set of patch bays. Each bay routes a distinct set of outputs to compatible inputs. These connections are implemented as a set of memory-mapped registers with a bit field for each input. The outputs are implemented as a set of bit encodings. Conceptually, a patch cord is used to connect an output to an input. In the SRU, a bit pattern that is associated with a signal output (shown as item 1 in Figure 5-9) is written to a bit field corresponding to a signal input.

The same encoding can be written to any number of bit fields in the same group. It is not possible to run out of patch points for an output signal.

Just as group A routes clock signals, each of the other groups route a collection of compatible signals. Group B routes serial data streams while group C routes frame sync signals. Group D routes signals to pins so that they may be driven off-chip. Note that all of the groups have an encoding that allows a signal to flow from a pin output to the input being specified by the bit field, but group D is required to route a signal to the pin input. Group F routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input. These groups are described in more detail in the following sections.

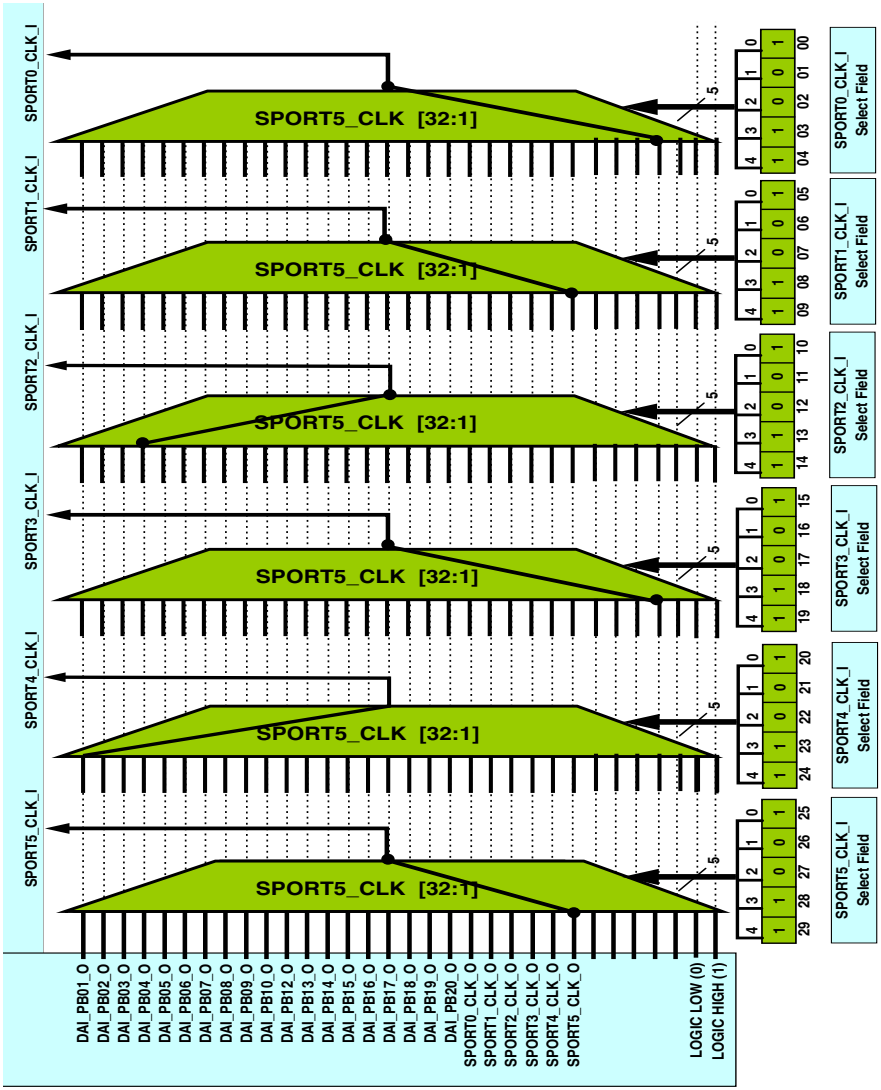


Figure 5-9. Example DAI SRU Group A Multiplexing (SRU_CLKx)

Rules for SRU Connections

There are two rules which apply to all routing:

1. Each input must connect to exactly one output
2. An output can feed any number of inputs

As an example:

- SPORT0_CLK_O is routed to SPORT1_CLK_I
- SPORT0_CLK_O is routed to SPORT2_CLK_I
- SPORT0_CLK_O is routed to SPORT3_CLK_I

Inputs may only be connected to outputs.

Making SRU Connections

In this section, three types of SRU routing are demonstrated.

In the first example, the SRU interconnects peripheral nodes internally (among a register group), as in SPORT1_CLK_O is routed to SPORT0_CLK_I to build an internal feedback. Note this junction does not affect the DAI pin buffers.

The second example uses the DAI pin buffer for signaling the clock off-chip, additionally a feedback to another SPORT peripheral is created within a DAI pin buffer. In this case, the same above example can be shown as:

SPORT1_CLK_O is routed to DAI_PB04_I
DAI_PB04_O is routed to SPORT0_CLK_I
SPORT1_CLK_PBEN_O is routed to PBEN04_I

This example requires three routs since the DAI pin buffer needs to be enabled. If it is not, the feedback junction is not effective.

Note that it is not possible to connect a signal in one group directly to a signal in a different group (analogous to wiring from one patch bay to another). However, group D is largely devoted to routing in this vein.

The third example routs a junction between 2 DAI pin buffers. Pin buffer 1 is an input and pin buffer 2 is an output. The routing is:

low level routed to DAI_PB01_I
low level routed to DAI_PBEN01_I
DAI_PB01_O routed to DAI_PB02_I
high level routed to DAI_PBEN02_I

This example can be enhanced with a condition with only a junction between both buffers if, for example, peripheral timer 0 has expired. In this case high level is routed to DAI_PBEN02_I changes to:
TIMER0_O routed to DAI_PBEN02_I

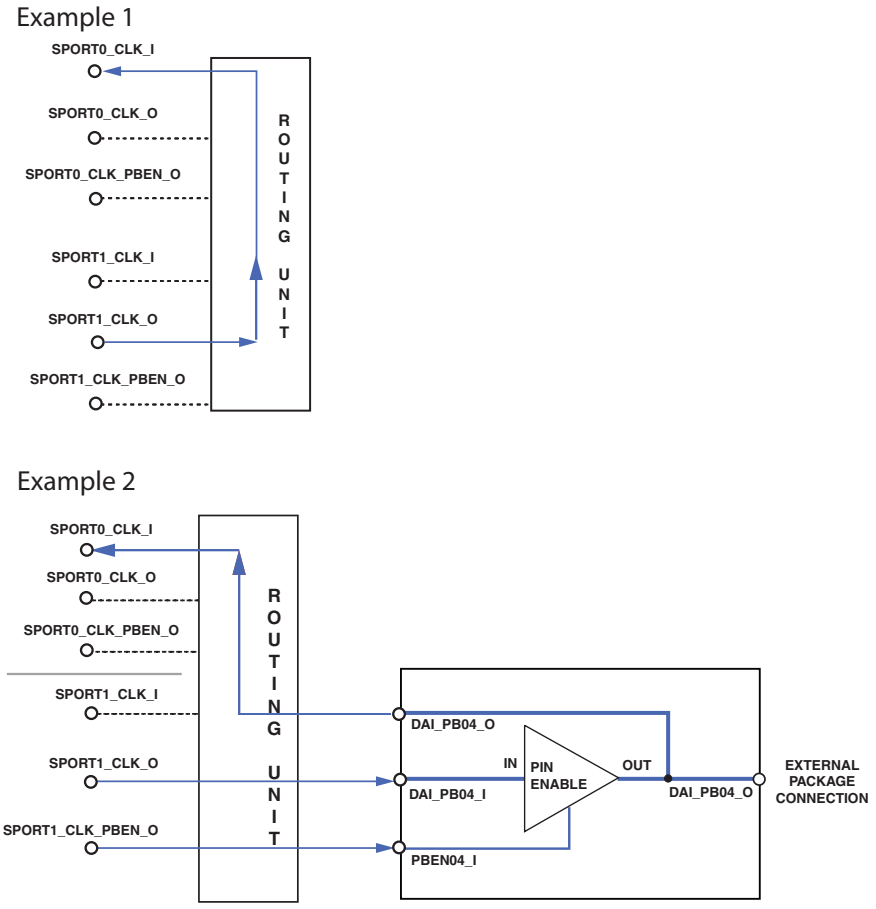
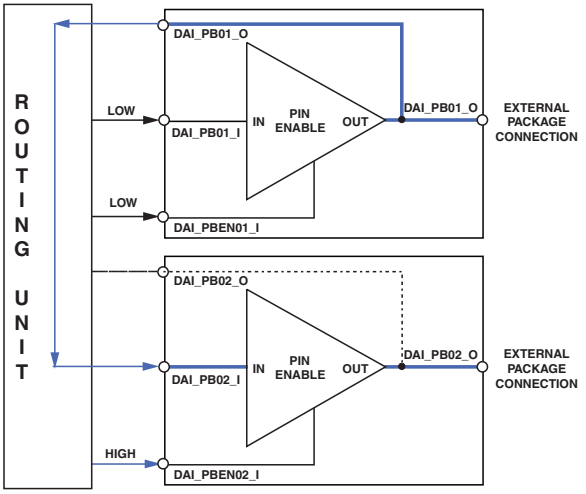


Figure 5-10. SRU Connection to SPORTs (Example 1 and 2)

Example 3



Example 4

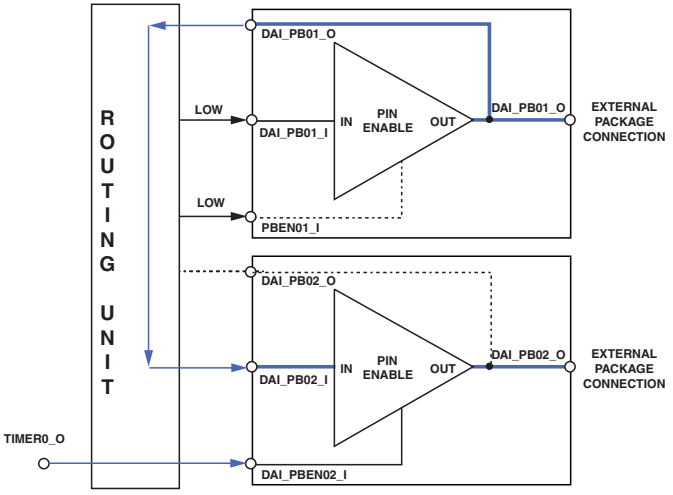


Figure 5-11. SRU Connection to SPORTs (Example 3 and 4)

Routing Capabilities

Table 5-1 provides an overview about thedifferent routing capabilities for the DAI unit. The DAI groups allow routing of specific signals like clocks, data, frame syncs.

Table 5-2. DAI Routing Capabilities

DAI Group	Input (xxxx_I)	Output (xxxx_O)	
A–Clocks	SPORT5–0 SRC3–0 IDP7–0 PCG AB (Ext CLK, Ext. Sync) S/PDIF TX (HFCLK, Ext. Sync) S/PDIF RX (Ext. CLK) SPIB	SPORT5–0 PCG AB S/PDIF RX (CLK, TDM CLK)	DAI Pin Buffer 20–1 Logic level high Logic level low
B–Data	SPORT5–0AB SRC5–0 IDP5–0 S/PDIF TX (data)	SPORT5–0AB SRC 3–0 S/PDIF RX	
C–Frame Sync	SPORT5–0 SRC3–0 IDP7–0	SPORT5–0 PCG AB S/PDIF TX (data) S/PDIF RX	
D–Pin Buffer Inputs	DAI Pin Buffer Input DAI Pin Buffer 19 Inversion DAI Pin Buffer 20 Inversion	SPORT5–0AB (data) SPORT5–0 (CLK, FS) TIMER2–0 S/PDIF RX (CLK, TDM CLK, FS, data) S/PDIF TX (data) S/PDIF RX (Ext. CLK) SPIB (CLK, data, CS) PDAP output strobe PCG AB (CLK, FS) SRC3–0 (data) FLAG15–10	

Table 5-2. DAI Routing Capabilities (Cont'd)

DAI Group	Input (xxxx_I)	Output (xxxx_O)	
E–Miscellaneous Signals	DAI Interrupt 31–22 FLAG15–10 TIMER2–0 SPIB (control, data) MISCA5–0 MISCA4 Input Inversion MISCA5 Input Inversion	TIMER2–0 PDAP output strobe PCG AB (CLK, FS) S/PDIF TX (block start)	DAI Pin Buffer 20–1 Logic level high Logic level low
F–Pin Buffer	DAI Pin Buffer Enable 20–1	SPORT5–0 AB (data) SPORT5–0 (CLK, FS) TIMER2–0 FLAG15–10 SPIB (control, data, CS) MISCA5–0	Logic level high Logic level low

Default Routing

When the processor comes out of reset, the SPORT junctions are bi-directional to the DAI pin buffers (Figure 5-12). This allows systems to use the SPORTs as either master or slave (without changing the routing scheme). Therefore, programs only need to use the SPORT control register settings to configure master or slave operation. Note that all DAI inputs which are not routed by default are tied to signal low.

Note that the default routing for the ADSP-2136x processors is different from previous SHARC families.

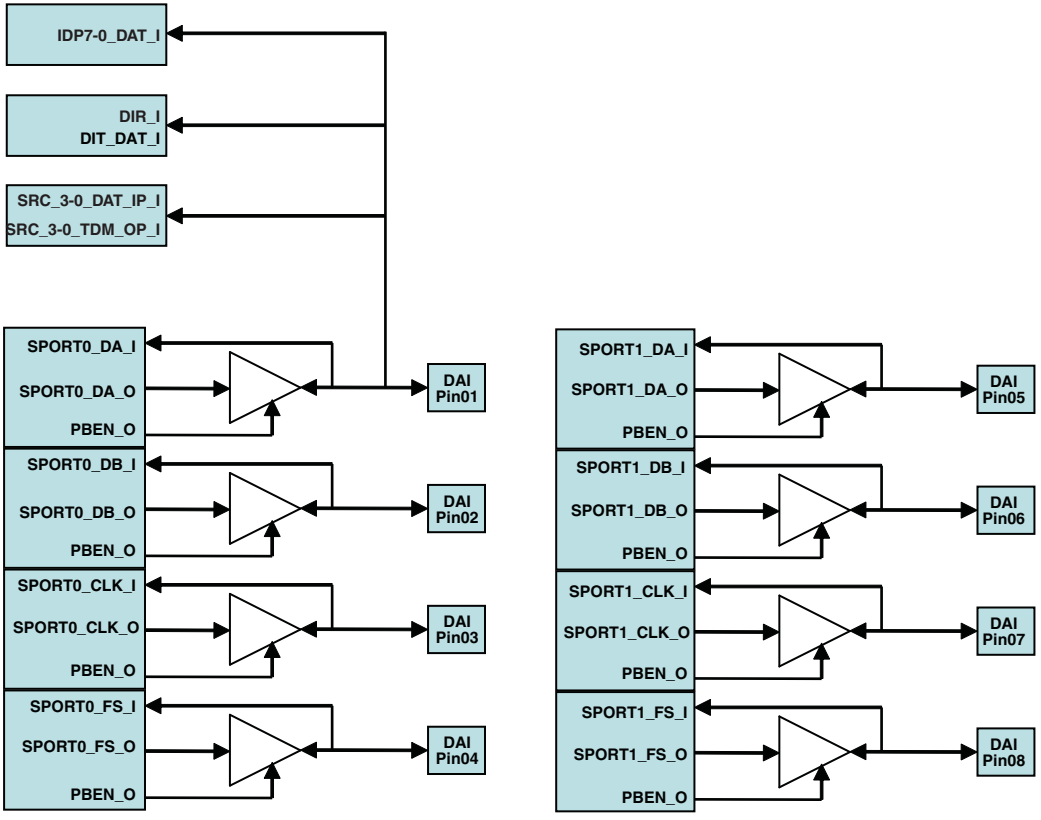


Figure 5-12. DAI Default Routing

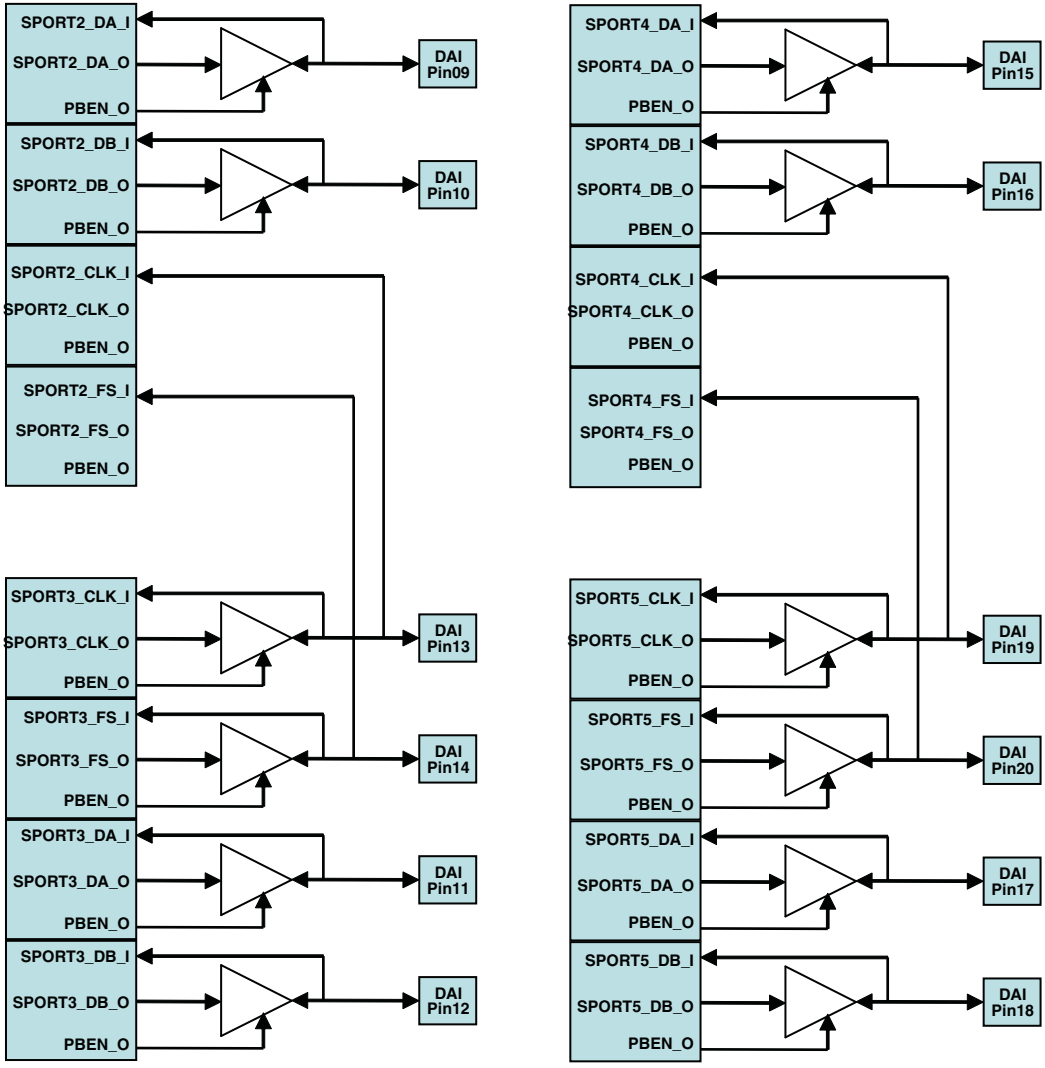


Figure 5-13. DAI Default Routing (con't)

Interrupt Controller

The DAI contains a dedicated interrupt controller that signals the core when DAI peripheral events occur.

System versus Exception Interrupts

Generally, interrupts are classified as system or exception. Exception events include any hardware interrupts (for example, resets) and emulation interrupts, math exceptions, and illegal accesses to memory that does not exist.

Programs can manage responses to signals by configuring registers. In a sample audio application, for example, upon detection of a change of protocol, the output can be muted. This change of output and the resulting behavior (causing the sound to be muted) results in an alert signal (an interrupt) being introduced in response (if the detection of a protocol change is a high priority interrupt).

Exception events are treated as high priority events. In comparison, system interrupts are "deterministic"—specific events emanating from a source (the causes), the result of which is the generation of an interrupt. The expiration of a timer can generate an interrupt, a signal that a serial port has received data that must be processed, a signal that an SPI has either transmitted or received data, and other software interrupts like the insertion of a trap that causes a breakpoint—all are conditions that tell the core that an event has occurred.

Since DAI specific events generally occur infrequently, the DAI interrupt controller classifies such interrupts as either high or low priority interrupts. Within these broad categories, programs can indicate which interrupts are high and which are classified as low.

Functional Description

There are several registers in the DAI interrupt controller that can be configured to control how the DAI interrupts are reported to and serviced by the core's interrupt controller.

The DAI contains its own interrupt controller that indicates to the core when DAI audio peripheral related events have occurred. Since audio events generally occur infrequently relative to the SHARC processor core, the DAI interrupt controller reduces all of its interrupts onto two interrupt signals within the core's primary interrupt systems.

Among other options, each DAI interrupt can be mapped either as a high or low priority interrupt in the primary interrupt controller. Certain DAI interrupts can be triggered on either the rising or the falling edge of the signals, and each DAI interrupt can also be independently masked.

Interrupt Channels

The DAI can handle up to 32 interrupts as shown below.

- 8 x IDP DMA channels (input data port)
- 2 x IDP FIFO status (input data port)
- 10 x miscellaneous interrupts (S/PDIF Tx, FLAGs)
- 8 x S/PDIF receiver status
- 4 x SRC (sample rate converter)

Interrupt Priorities

As described above, the DAI interrupt controller registers provide 32 independently-configurable interrupts labeled `DAI_INT31-0`.

Just as the core has its own interrupt latch registers (IRPTL and LIRPTL), the DAI has its own latch registers (DAI_IRPTL_L and DAI_IRPTL_H). When a DAI interrupt is configured to be high priority, it is latched in the DAI_IRPTL_H register. When any bit in the DAI_IRPTL_H register is set (= 1), bit 11 in the IRPTL register is also set and the core services that interrupt with high priority. When a DAI interrupt is configured to be low priority, it is latched in the DAI_IRPTL_L register. Similarly, when any bit in the DAI_IRPTL_L register is set (= 1), bit 6 in the LIRPTL register is also set and the core services that interrupt with low priority.

By default, interrupts are mapped onto a low priority interrupt.

Miscellaneous Interrupts

As described above, the DAI interrupt controller registers provide 10 independently-configurable interrupts labeled as SRU_MISCx_INT. Any trigger on the DAI inputs DAI_INTx_I can cause an interrupt latch event in SRU_MISCx_INT if enabled.

Signals from the SRU can also be used to generate interrupts. For example, when SRU_EXTMISCA2_INT (bit 30) of DAI_IRPTL_H is set (= 1), any signals from the external miscellaneous channel 2 generate an interrupt, the DAI interrupts trigger an interrupt in the core and the interrupt latch is set. A read of this bit does not reset it to zero. The bit is only set to zero when the cause of the interrupt is cleared. A DAI interrupt indicates the source (in this case, external miscellaneous A, channel 2), and checks the IVT for an instruction (next operation) to perform.

Core versus DAI Interrupts

In the ADSP-2136x processor, a pair of registers (DAI_IRPTL_H and DAI_IRPTL_L) replace functions normally performed by the IRPTL register. A single register (DAI_IRPTL_PRI) specifies to which latch these interrupts are mapped.

Two registers (DAI_IRPTL_RE and DAI_IRPTL_FE) replace the DAI peripheral's version of the IMASK register. As with the IMASK register, these DAI registers provide a way to specify which interrupts to notice and handle, and which interrupts to ignore. These dual registers function like the IMASK register, but with a higher degree of granularity.

The DAI interrupt controller has the same 6 cycle latency to respond to asynchronous interrupts as the core interrupt controller.

Note that the IRPTL and LIRPTL registers are system registers. All DAI interrupt registers (DAI_IRPTL_x) are memory-mapped registers.

Interrupt Events

For interrupt sources that correspond to waveforms (as opposed to DAI event signals such as DMA complete or buffer full), the edge of a waveform may be used as an interrupt source as well. Just as interrupts can be generated by a source, interrupts can also be generated and latched on the rising (or falling) edges of a signal.

Only the DAI interrupt controller latches interrupts on both edges. This ability does not exist in the core interrupt controller.

Use of the DAI_IRPT_RE or DAI_IRPT_FE registers allows programs to notice and respond to rising edges, falling edges, both rising and falling edges, or neither rising nor falling edges so they can be masked separately.

When a signal comes in, the system needs to determine what kind of signal it is and what kind of protocol, as a result, to service. The preamble indicates the signal type. When the protocol changes, output (signal) type is noted.

For audio applications, the processor needs information about interrupt sources that correspond to waveforms (not event signals). As a result, the

falling edge of the waveform may be used as an interrupt source as well. Programs may select any of these three conditions:

- Latch on the rising edge
- Latch on the falling edge
- Latch on *both* the rising and falling edge

Table 5-3 shows which interrupts are valid on rising and or falling edges.

Table 5-3. Interrupt Valid Edges

Interrupt Source	DAI_IRPTL_RE	DAI_IRPTL_FE
S/PDIF Rx	Yes	Yes
IDP_FIFO	Yes	No
IDP_DMA	Yes	No
SRC Mute	Yes	Yes
Miscellaneous	Yes	Yes

Enabling responses to changes in conditions of signals (including changes in DMA state, introduction of error conditions, and so on) can only be done using the `DAI_IRPT_RE` register.

Servicing Interrupts

Any asynchronous or synchronous interrupt causes has latency, since it forces the core to stop processing an instruction in process, then vector to the interrupt service routine (ISR, which is basically an interrupt vector table lookup), then proceed to implement the instruction referenced in the IVT. For more information, see Chapter B, Interrupts.

When an interrupt from the DAI must be serviced, one of the two core ISRs must query the DAI's interrupt controller to determine the source(s). Sources can be any one or more of the interrupt controller's 32

configurable channels (DAI_INT[31:0]). When DAI_IRPTL_H is read, the high priority latched interrupts are cleared. When DAI_IRPTL_L is read, the low priority latched interrupts are cleared. For more information, see "DAI Interrupt Controller Registers" on page A-77.

The DAI triggers two interrupts in the primary IVT—one each for low or high priority. When any interrupt from the DAI needs to be serviced, one of the two core ISRs must interrogate the DAI's interrupt controller to determine the source(s).

Reading the DAI's interrupt latches clears the interrupts. Therefore, the ISR must service *all* the interrupt sources it discovers. That is, if multiple interrupts are latched in one of the DAI_IRPTL_x registers, all of them must be serviced before executing an RTI instruction. For more information, see "DAI Interrupt Controller Registers" on page A-77.

Debug Features

The following sections describe features that can be used to help in debugging the DAI.

Shadow Registers

The interrupt service routine (ISR) must read the DAI_IRPTL_H or DAI_IRPTL_L register to know all the interrupts currently latched. The DAI_IRPTL_H register is for high priority interrupts and the DAI_IRPTL_L register is for low priority interrupts. Reads of these registers clears the latched interrupt bits.

The shadow registers DAI_IRPTL_L_SH and DAI_IRPTL_H_SH are provided for registers DAI_IRPTL_L and DAI_IRPTL_H respectively. Reads of the shadow registers returns the data in the DAI_IRPTL_L and DAI_IRPTL_H registers respectively without clearing the contents of these registers.

Loop Back Routing

The serial peripherals (SPORT and SPI) support an internal loop back mode. If the loop back bit for each peripheral is enabled, it connects the transmitter with the receiver block internally (does not signal off-chip). The SRU can be used for this propose. Table 5-4 describes the different possible routings based on the peripheral.

The peripheral's loop back mode for debug is independent from both of the signal routing units.

Table 5-4. Loop Back Routing

Peripheral	Loopback Mode	SRU Internal Routing for Loopback	SRU External Routing for Loopback
IDP	N/A	N/A	N/A
SPORT	Yes	SPORTx_xx_O → SPORTx_xx_I	SPORTx_xx_O → DAI_PBxx_I DAI_PBxx_O → SPORTx_xx_I
S/PDIF Tx/Rx	No	DIT_O → DIR_I	DIT_O → DAI_PBxx_I DAI_PBxx_O → DIR_I
Timer	No	TIMERx_O → TIMERx_I	TIMERx_O → DAI_PBxx_I DAI_PBxx_O → TIMERx_I
SPI	Yes	No	SPIx_xx_O → DAI_PBxx_I DAI_PBxx_O → SPIx_xx_I

Programming Model

As discussed in the previous sections, the signal routing unit is controlled by writing values that correspond to signal sources into bit fields that further correspond to signal inputs. The SRU is arranged into functional groups such that the registers that are made up of these bit fields accept a common set of source signal values.

In order to ease the coding process, the header file `SRU.H` is included with the VisualDSP++ tools. This file implements a macro that abstracts away most of the work of signal assignments and functions. The macro has identical syntax in C/C++ and assembly, and makes a single connection from an output to an input as shown below.

```
SRU(Output Signal, Input Signal);
```

The names passed to the macro are the names given in "DAI Signal Routing Unit Registers" on page A-80. To use this macro, add the following line to your source code as shown in Listing 5-1:
#include <sru.h>;

Listing 5-1. DAI Macro Code

```
#include <sru.h>;
/* The following lines illustrate how the macro is used: */
/* Route SPORT 1 clock output to pin buffer 5 input */
       SRU(SPORT1_CLK_O,DAI_PB05_I);

   /* Route pin buffer 14 out to IDP3 frame sync input */
       SRU(DAI_PB14_O,IDP3_FS_I);

   /* Connect pin buffer enable 19 to logic low */
       SRU(LOW,PBEN19_I);
```

Additional example code is available on the Analog Devices Web site.

There is a macro that has been created to connect peripherals used in a DAI configuration. This code can be used in both assembly and C code. See the `INCLUDE` file `SRU.H`.

There is also a software plug-in called the *Expert DAI* that greatly simplifies the task of connecting the signals described in this chapter. This plug-in is described in Engineer-to-Engineer Note EE-243, "Using the Expert DAI for ADSP-2126x and ADSP-2136x SHARC Processors". This EE note is also found on the Analog Devices Web site.

6 SERIAL PORTS

The ADSP-2136x processors have six independent, synchronous serial ports (SPORTs) that provide an I/O interface to a wide variety of peripheral devices. They are called SPORT0, SPORT1, SPORT2, SPORT3, SPORT4, and SPORT5. Each SPORT has its own set of control registers and data buffers. With a range of clock and frame synchronization options, the SPORTs allow a variety of serial communication protocols and provide a glueless hardware interface to many industry-standard data converters and codecs.

Serial ports offer the additional features and capabilities shown in Table 6-1, Figure 6-1, and described in the following list.

Table 6-1. Serial Port Feature Summary

Feature	SPORT5–0[AB]
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	Yes
Interrupt Default Routing	Yes (P3I–P8I)
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes
Transmission Full Duplex	No

Table 6-1. Serial Port Feature Summary (Cont'd)

Feature	SPORT5–0[AB]
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	2 per SPORT
DMA Chaining	Yes
Interrupt Source	Core/DMA
Boot Capable	No
Local Memory	No
Clock Operation	PCLK/4

Bidirectional (transmit or receive) functions provide greater flexibility for serial communications. Serial ports can operate at a maximum of one-fourth the peripheral clock rate of the processor. If channels A and B are active, each SPORT has a maximum throughput of 2 x PCLK/4 rate.

Serial port data can be automatically transferred to and from on-chip and from off-chip memory using DMA block transfers. In addition to standard synchronous serial mode, each SPORT offers a time division multiplexed (TDM) multichannel mode, left-justified mode, and I^2S mode.

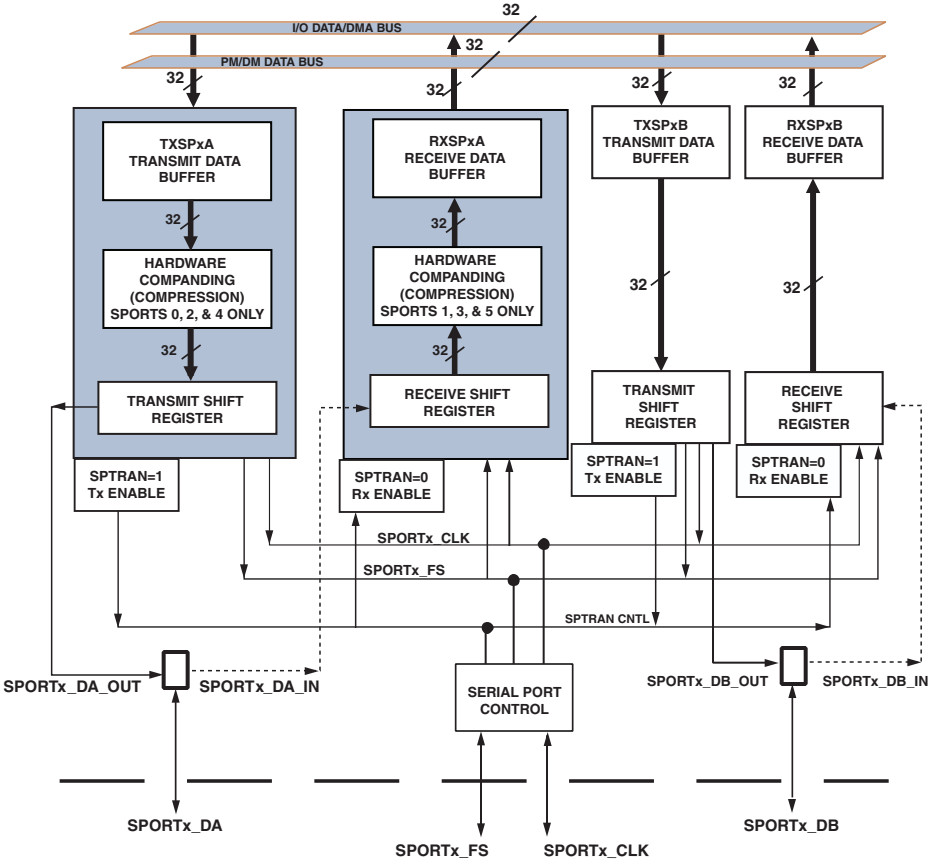


Figure 6-1. Serial Port Block Diagram

Features

Serial ports offer the following features and capabilities:

- Four operation modes ("Selecting Operating Modes" on page 6-18):

1. Standard serial
2. Left-justified
3. I^2S
4. Multichannel

- Two bidirectional channels (A and B) per serial port, configurable as either transmitters or receivers. Each serial port can also be configured as two receivers or two transmitters, permitting two unidirectional streams into or out of the same serial port. This bidirectional functionality provides greater flexibility for serial communications. Also, two SPORTs can be combined to enable full-duplex, dual-stream communications.
- All serial data signals have programmable receive and transmit functions and thus have one transmit and one receive data buffer register (double-buffer) and a bidirectional shift register associated with each serial data signal. Double-buffering provides additional time to service the SPORT.
- An internally-generated serial clock and frame sync provide signals in a wide range of frequencies. Alternately, the SPORT can accept clock and frame sync input from an external source.
- Interrupt-driven, single word transfers to and from on-chip memory controlled by the processor core, described in "Single Word Transfers" on page 6-47.

- DMA transfers to and from on-chip and off-chip memory. Each SPORT can automatically receive or transmit an entire block of data both on- and off-chip.
- Chained DMA operations for multiple data blocks, see "DMA Chaining" on page 2-20.
- DMA Chain insertion mode allows the SPORTs to change DMA priority during chaining, see "Enter DMA Chain Insertion Mode" on page 6-57.
- Data words between 3 and 32 bits inlength, either most significant bit (MSB) first or least significant bit (LSB) first. Words must be between 8 and 32 bits in length for I^2S and left-justified mode.
- 128-channel TDM is supported in multichannel mode operation, useful for H.100/H.110 and other telephony interfaces described in "Multichannel Operation" on page 6-40.
- µ-law and A-law compression/decompression hardware companding on transmitted and received words when the SPORT operates in TDM mode.

Receive comparison and 2-dimensional DMA are not supported in the ADSP-2136x processor.

Pin Descriptions

Table 6-2 describes pin function.

Table 6-2. SPORT Pin Descriptions

Internal Nodes	Direction	Description
SPORT5–0_DA_I/O	I/O	**Data Receive or Transmit Channel A.** Bidirectional data pin. This signal can be configured as an output to transmit serial data, or as an input to receive serial data.
SPORT5–0_DB_I/O	I/O	**Data Receive or Transmit Channel B.** Bidirectional data pin. This signal can be configured as an output to transmit serial data, or as an input to receive serial data.
SPORT5–0_CLK_I/O	I/O	**Transmit/Receive Serial Clock.** This signal can be either internally or externally generated.
SPORT5–0_FS_I/O	I/O	**Transmit/Receive Frame Sync.** The frame sync pulse initiates shifting of serial data. This signal is either generated internally or externally. It can be active high or low or an early or a late frame sync, in reference to the shifting of serial data.
SPORT5–0_DA_PBEN_O	O	Only driven in master mode.
SPORT5–0_DB_PBEN_O	O	
SPORT5–0_CLK_PBEN_O	O	
SPORT5–0_FS_PBEN_O	O	

SRU Configuration

Any of the serial port's signals can be mapped to digital applications interface (DAI_Px) pins through the signal routing unit (SRU) as shown in Table 6-3. For more information, see Chapter 5, Digital Application Interface.

Table 6-3. SPORT DAI/SRU Signal Connections

Internal Node	DAI Connection	SRU Register
Inputs		
SPORT5–0_CLK_I	Group A	SRU_CLK1–0
SPORT5–0_FS_I	Group C	SRU_FS0
SPORT5–0_DA_I	Group B	SRU_DAT2–0
SPORT5–0_DB_I		
Outputs		
SPORT5–0_CLK_O	Group A, D	
SPORT5–0_FS_O	Group C, D	
SPORT5–0_DA_O	Group B, D	
SPORT5–0_DB_O		
SPORT5–0_CLK_PBEN_O	Group F	
SPORT5–0_FS_PBEN_O		
SPORT5–0_DA_PBEN_O		
SPORT5–0_DB_PBEN_O		

SRU SPORT Receive Master

If the SPORT is operating as receive master, it must feed its master output clock back to its input clock. This is required to trigger the SPORT's state machine. Using SPORT 4 as an example receive master, programs should route `SPORT4_CLK_O` to `SPORT4_CLK_I`. This is not required if the SPORT is operating as a transmitter in master mode.

SRU SPORT Signal Integrity

There is some sensitivity to noise on the clock (`SPORTx_CLK`) and frame sync (`SPORTx_FS`) signals when the SPORT is configured as a master receiver. By correctly programming the signal routing unit (SRU) clock

and frame sync registers, the reflection sensitivity in these signals can be avoided.

Figure 5-10 on page 5-18 shows the default routing of the serial port where the SRU maps:

- the signal from the DAI pin (DAI_PBxx_O) back to the SPORT clock input (SPORTx_CLK_I)
- the SPORT clock output (SPORTx_CLK_O) to the pin buffer input (DAI_PBxx_I)

By redirecting the signals as shown in Figure 6-2 where the clock and frame sync outputs are routed directly back to their respective inputs, the signal sensitivity issue can be avoided.

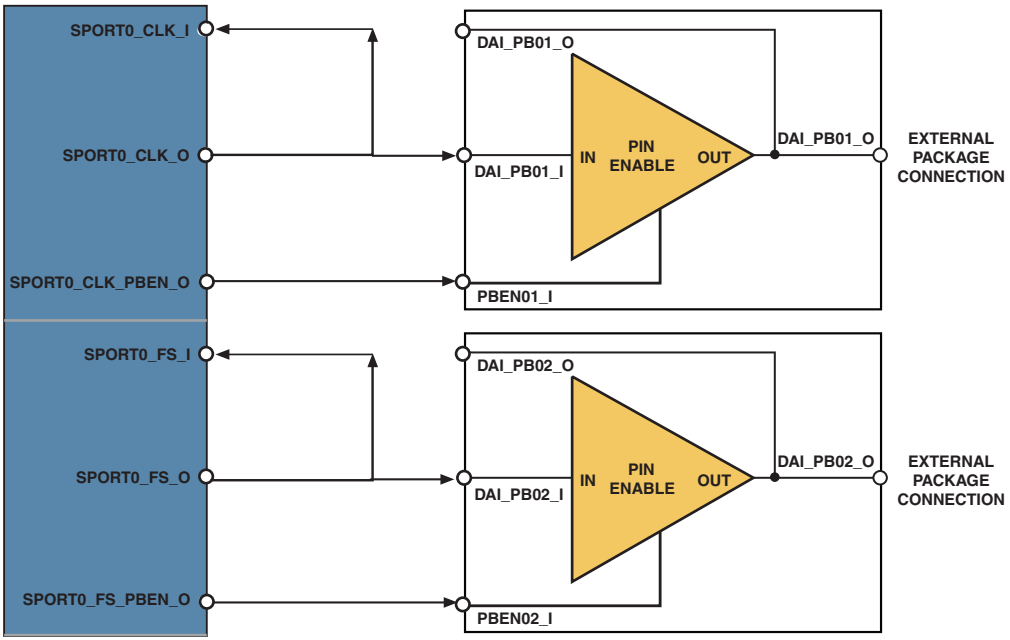


Figure 6-2. SRU Configuration When SPORT is Master Receiver.

Functional Description

A serial port receives serial data on one of its bidirectional serial data signals configured as inputs, or transmits serial data on the bidirectional serial data signals configured as outputs. It can receive or transmit on both channels simultaneously and unidirectionally, where the pair of data signals can both be configured as either transmitters or receivers.

The SPORTx_DA and SPORTx_DB channel data signals on each SPORT cannot transmit and receive data simultaneously for full-duplex operation. Two SPORTs must be combined to achieve full-duplex operation. The SPTRAN bit in the SPCTLx register controls the direction for both the A and B channel signals.

The data direction of channel A and channel B on a particular SPORT must be the same.

Serial communications are synchronized to a clock signal. Every data bit must be accompanied by a clock pulse. Each serial port can generate or receive its own clock signal (SPORTx_CLK). Internally-generated serial clock frequencies are configured in the DIVx registers. The A and B channel data signals shift data based on the rate of SPORTx_CLK.

In addition to the serial clock signal, data may be signaled by a frame synchronization signal. The framing signal can occur at the beginning of an individual word or at the beginning of a block of words. The configuration of frame sync signals depends upon the type of serial device connected to the processor. Each serialport can generate or receive its own frame sync signal (SPORTx_FS) for transmitting or receiving data. Internally-generated frame sync frequencies are configured in the DIVx registers. Both the A and B channel data signals shift data based on their corresponding SPORTx_FS signal.

Figure 6-1 on page 6-3 shows a block diagram of a serial port. Setting the SPTRAN bit enables the data buffer path, which, once activated, responds by shifting data in response to a frame sync at the rate of SPORTx_CLK. An

application program must use the correct serial port data buffers, according to the value of SPTRAN bit. The SPTRAN bit enables either the transmit data buffers for the transmission ofA and B channel data, or it enables the receive data buffers for the reception of A and B channel data. Inactive data buffers are not used.

When programming the serial port channel (A or B) as a transmitter, only the corresponding transmit buffers TXSPxA and TXSPxB become active, while the receive buffers (RXSPxA and RXSPxB) remain inactive. Similarly, when SPORT channels A and B are programmed to receive, only the corresponding RXSPxA and RXSPxB buffers are activated.

The processor's SPORTs are not UARTs and cannot communicate with an RS-232 device or any other asynchronous communications protocol. One way to implement RS-232 compatible communication with the processor is to use two of the FLAG pins as asynchronous data receive and transmit signals.

Registers

The ADSP-2136x processor has six serial ports. Each SPORT has two data paths corresponding to channel A and channel B.

The registers used to control and configure the serial ports are part of the IOP register set. Each SPORT has its own set of 32-bit control registers and data buffers.

The main control register for each serial port is the serial port control register, SPCTLx. These registers are described in "Serial Port Registers" on page A-30.

When changing operating modes, clear the serial port control register before the new mode is written to the register.

The SPCTLx registers control the operating modes of the serial ports for the I/O processor. Table 6-4 lists all the bits in the SPCTLx register. Note that

the shaded cells denote that the bits have same function in all operating modes.

Table 6-4. SPCTLx Control Bit Comparison

Bit	Standard Serial Mode	I^2S and Left-Justified Mode	Multichannel Mode	
			Transmit Control Bits (SPORT0, 2, 4)	Receive Control Bits (SPORT1, 3, 5)
0	SPEN_A	SPEN_A	Reserved	Reserved
1 DT	YPE	Reserved	DTYPE	DTYPE
2 DT	YPE	Reserved	DTYPE	DTYPE
3 LSB	F	Reserved	LSBF	LSBF
4	SLEN0	SLEN0	SLEN0	SLEN0
5	SLEN1	SLEN1	SLEN1	SLEN1
6	SLEN2	SLEN2	SLEN2	SLEN2
7	SLEN3	SLEN3	SLEN3	SLEN3
8	SLEN4	SLEN4	SLEN4	SLEN4
9	PACK	PACK	PACK	PACK
10 ICL	K	MSTR	Reserved	ICLK
11	OPMODE	OPMODE	OPMODE	OPMODE
12 CKRE		Reserved	CKRE	CKRE
13 FS	R	Reserved	Reserved	Reserved
14 IFS		Reserved	Reserved	IMFS
15 DIFS		DIFS	Reserved	Reserved
16 LFS		L_FIRST	LTDV	LMFS
17 LAF	S	LAFS	Reserved	Reserved
18	SDEN_A	SDEN_A	SDEN_A	SDEN_A
19	SCHEN_A	SCHEN_A	SCHEN_A	SCHEN_A
20	SDEN_B	SDEN_B	SDEN_B	SDEN_B

Table 6-4. SPCTLx Control Bit Comparison (Cont'd)

Bit	Standard Serial Mode	I^2S and Left-Justified Mode	Multichannel Mode	
			Transmit Control Bits (SPORT0, 2, 4)	Receive Control Bits (SPORT1, 3, 5)
21	SCHEN_B	SCHEN_B	SCHEN_B	SCHEN_B
22 FS_BO TH		Reserved	Reserved	Reserved
23	BHD	BHD	BHD	BHD
24 SP EN_B		SPEN_B	Reserved	Reserved
25 SPTRAN		SPTRAN	Reserved	Reserved
26 D ERR_B		DERR_B	TUVF_B	ROVF_B
27 D XS_B		DXS_B	TXS_B	RXS_B
28 D XS_B		DXS_B	TXS_B	RXS_B
29 D ERR_A		DERR_A	TUVF_A	ROVF_A
30 D XS_A		DXS_A	TXS_A	RXS_A
31 D XS_A		DXS_A	TXS_A	RXS_A

Control Registers (SPCTLx)

The SPCTLx registers control serial port modes and are part of the SPCTLx (transmit and receive) control registers. Other bits in these registers set up DMA and I/O processor-related serial port features. For information about configuring a specific operation mode, refer to Table 6-5 on page 6-19 and "Operating Modes" on page 6-34.

Multichannel Control Registers (SPMCTLxy)

There is one global control and status register for each SPORT pair (0 and 1, 2 and 3, 4 and 5) for multichannel operation. These registers define the number of channels, provide the status of the current channel, enable multichannel operation, and set the multichannel frame delay. These

registers are described in "SPORT Multichannel Control Registers (SPMCTLxy)" on page A-43.

Data Buffers

When programming the serial port channel (A or B) as a transmitter, only the corresponding TXSPxA and TXSPxB buffers become active while the receive buffers RXSPxA and RXSPxB remain inactive. Similarly, when the SPORT channel A and B are programmed as receive-only the corresponding RXSPxA and RXSPxB are activated. Do not attempt to read or write to inactive data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, unpredictable results may occur.

Word lengths of less than 32 bits are automatically right-justified in the receive and transmit buffers.

Transmit Buffers (TXSPxA/B)

The transmit buffers (TXSP5-0A, TXSP5-0B) are the 32-bit transmit data buffers for SPORT5–0 respectively. These buffers must be loaded with the data to be transmitted if the SPORT is configured to transmit on the A and B channels. The data is loaded automatically by the DMA controller or loaded manually by the program running on the processor core.

The transmit buffers act like a two-location FIFO because they have a data register plus an output shift register. Two 32-bit words may both be stored in the transmit queue at any one time. When the transmit register is loaded and any previous word has been transmitted, the register contents are automatically loaded into the output shifter. An interrupt occurs when the output transmit shifter has been loaded, signifying that the transmit buffer is ready to accept the next word (for example, the transmit buffer is not full). This interrupt does not occur when serial port DMA is enabled or when the corresponding mask bit in the LIRPTL/IRPTL register is set.

Transmit Path

If the serial port is configured as a serial transmitter, the data transmitted is written to the TXSPxA/TXSPxB buffer. The data is (optionally) companded in hardware on the primary A channel (SPORT 0, 2, and 4 only), then automatically transferred to the transmit shift register, because companding is not supported on the secondary B channels. The data in the shift register is then shifted out via the SPORT's SPORTx_DA or SPORTx_DB signal, synchronous to the SPORTx_CLK clock. If framing signals are used, the SPORTx_FS signal indicates the start of the serial word transmission.

The SPORTx_DA or SPORTx_DB signal is always driven if the serial port is enabled as transmitter (SPEN_A or SPEN_B = 1 in the SPCTLx control register), unless it is in multichannel mode and an inactive time slot occurs.

When the SPORT is configured as a transmitter (SPTRAN = 1), the TXSPxA and TXSPxB buffers, and the channel transmit shift registers respond to SPORTx_CLK and SPORTx_FS to transmit data. The receive RXSPxA and RXSPxB buffers, and the receive shift registers are inactive and do not respond to SPORTx_CLK and SPORTx_FS signals. Since these registers are inactive, reading from an empty buffer causes the core to hang indefinitely.

If the SPORTs are configured as transmitters (SPTRAN bit = 1 in SPCTL), programs should not read from the inactive RXSPxA and RXSPxB buffers. This causes the core to hang indefinitely since the receive buffer status is always empty.

Receive Buffers (RXSPxA/B)

The receive buffers (RXSP5-0A, RXSP5-0B) are the 32-bit receive data buffers SPORT5–0 respectively. These 32-bit buffers become active when the SPORT is configured to receive data on the A and B channels. When a SPORT is configured as a receiver, the RXSPxA and RXSPxB registers are automatically loaded from the receive shifter when a complete word has

been received. The data is then loaded to internal memory by the DMA controller or read directly by the program running on the processor core.

Receive Path

If the serial data signal is configured as a serial receiver (SPTRAN = 0), the receive portion of the SPORT shifts in data from the SPORTx_DA or SPORTx_DB signal, synchronous to the SPORTx_CLK receive clock. If framing signals are used, the SPORTx_FS signal indicates the beginning of the serial word being received. When an entire word is shifted in on the primary A channel, the data is (optionally) expanded (SPORT1, 3, and 5 only), then automatically transferred to the RXSPxA buffer. When an entire word is shifted in on the secondary channel, it is automatically transferred to the RXSPxB buffer.

When the SPORT is configured as a receiver (SPTRAN = 0), the RXSPxA and RXSPxB buffers, and the channel receive shift registers respond to SPORTx_CLK and SPORTx_FS for reception of data. The transmit TXSPxA and TXSPxB buffer registers and transmit A and B shift registers are inactive and do not respond to the SPORTx_CLK and SPORTx_FS. Since the TXSPxA and TXSPxB buffers are inactive, writing to a transmit data buffer causes the core to hang indefinitely.

When the SPORT is configured as a receiver (SPTRAN = 0), the receive buffers are activated. The receive buffers act like a three-location FIFO because they have two data registers plus an input shift register.

Buffer Status

Serial ports provide status information about data buffers via the DXS_A and DXS_B status bits and error status via the DERR_x bits in the SPCTL register. Depending on the SPTRAN setting, these bits reflect the status of either the TXSPxy or RXSPxy data buffers.

Bits 31–30 (RXS_A) and bits 28–27 (RXS_B) in the SPCTLx registers indicate the status of the channel's receive buffer contents as follows: 00 = buffer empty, 01 = reserved, 10 = buffer partially full, 11 = buffer full.

If your program causes the core processor to attempt to read from an empty receive buffer or to write to a full transmit buffer, the access is delayed until the buffer is accessed by the external I/O device. This delay is called a core processor hang. If you do not know if the core processor can access the receive or transmit buffer without a hang, the buffer's status should be read first (in SPCTLx) to determine if the access can be made.

The status bits in SPCTLx are updated during reads and writes from the core processor even when the serial port is disabled. Programs should disable the serial port when writing to the receive buffer or reading from the transmit buffer.

When the SPORT is configured as a transmitter, the DERR_x bits provide transmit underflow status. As a transmitter, if FSR = 1, the DERR_x bits indicate whether the SPORTx_FS signal (from an internal or external source) occurred while the DXS buffer was empty. If FSR = 0, DERR_x is set whenever the SPORT is required to transmit and the transmit buffer is empty. The SPORTs transmit data whenever they detect a SPORTx_FS signal.

- 0 = No SPORTx_FS signal occurred while TXSPxA/B buffer is empty.
- 1 = SPORTx_FS signal occurred while TXSPxA/B buffer is empty.

When the SPORT is configured as a receiver, the DERR_x bits provide receive overflow status. As a receiver, it indicates when the channel has received new data while the RXS_A buffer is full. New data overwrites existing data.

- 0 = No new data while RXSPxA/B buffer is full.
- 1 = New data while RXSPxA/B buffer is full.

Two complete 32-bit words can be stored in the receive buffer while a third word is being shifted in. The third word overwrites the second if the first word has not been read out (by the processor core or the DMA controller). When this happens, the receive overflow status bit is set in the serial port control register. Almost three complete words can be received without the receive buffer being read before an overflow occurs. The overflow status is generated on thelast bit of the third word. The`DERR_x` status bits are sticky and are cleared only by disabling the serial port.

If the SPORTs are configured as receivers (`SPTRAN` bit = 0 in `SPCTLx`), programs should not read from the inactive `TXSPxA` and `TXSPxB` buffers. If the core keeps writing to the inactive buffer, the transmit buffer status becomes full. This causes the core to hang indefinitely since data is never transmitted out of the deactivated transmit data buffers.

The data independent frame sync bit (`DIFS`) allows SPORTs to generate frame syncs regardless of the receive or transmit buffer status for non-multichannel modes.

Multichannel Buffer Status

In multichannel mode, the `DXS_x` and `DERR_x` bits are redefined due to the fixed-directional functionality of the `SPCTLx` registers. When the `SPCTL1`, `SPCTL3`, and `SPCTL5` registers are configured for multichannel mode, the receive overflow bit, `ROVF_x`, indicates when the specified channel has received new data while the `RXS_x` buffer is full.

Similarly, when the `SPCTL0`, `SPCTL2`, and `SPCTL4` registers are configured for multichannel mode, the transmit overflow bit, (`TUVF_x`), indicates that a new frame sync signal, (`SPORT0_FS`/`SPORT2_FS`/`SPORT4_FS`), was generated while the `TXS_x` buffer was empty.

The ROVF_x or TUVF_x overflow/underflow status bit in the SPCTLx register becomes fixed in multichannel mode only as either the ROVF_x overflow status bit (SPORTs 1, 3, and 5) or TUVF_x underflow status bit (SPORTs 0, 2, and 4).

Selecting Operating Modes

The SPORTs operate in four modes:

- Standard serial mode, described in "Standard Serial Mode" on page 6-34
- Left-justified mode, described in "Left-Justified Mode" on page 6-36
- I^2S mode, described in "I2S Mode" on page 6-38
- Multichannel mode, described in "Multichannel Operation" on page 6-40

Bit names and their functionality change based on the SPORT operating mode. See the mode specific section for the bit names and their functions.

Pairings of SPORTs (0 and 1, 2 and 3, 4 and 5) are only used in multichannel mode and loopback mode for testing.

In standard serial, left-justified and I^2S modes, (when both A and B channels are used), the channels transmit or receive data simultaneously. The channels send or receive bit 0 on the same edge of the serial clock, bit 1 on the next edge of the serial clock, and so on.

Mode Selection

The serial port operating mode can be selected via the SPCTLx and the SPMCTLxy registers. "Serial Port Registers" on page A-30.

1. The operating mode bit 11(OPMODE) of the SPCTLx register selects between I^2S, left-justified, and standard serial/multichannel mode.
2. Bit 17 of the SPCTLx register selects between I^2S mode and left-justified mode.
3. In multichannel mode, the bit 0 (MCEA) in the SPMCTLxy register enables the A channels and the bit 23 (MCEB) in the SPMCTLxy register enables the B channels.

The SPCTLx register is unique in that the name and functionality of its bits changes depending on the operation mode selected. In each section that follows, the bit names associated with the operating modes are described. Table 6-5 provides values for each of the bits in the SPORT serial control (SPCTLx) registers that must be set in order to configure each specific SPORT operation mode. An X in a field indicates that the bit is not supported for the specified operating mode. The shaded columns indicate that the bits come from different control registers.

Table 6-5. SPORT Operation Modes

OPERATING MODES	SPCTLx Bits			SPMCTLxy bits	
	Bit 11	Bit 17	Bit 16	MCEA	MCEB
Standard serial mode	0	0, 1	X	0	0
I^2S (Tx/Rx on Left Channel First)	1	0	1	0	0
I^2S (Tx/Rx on Right Channel First)	1	0	0	0	0
Left-Justified Mode (Tx/Rx on FS Rising Edge)	1	1	0	0	0
Left-Justified (Tx/Rx on FS Falling Edge)	1	1	1	0	0

Table 6-5. SPORT Operation Modes (Cont'd)

OPERATING MODES	SPCTLx Bits			SPMCTLxy bits	
	Bit 11	Bit 17	Bit 16	MCEA	MCEB
Multichannel A Channels	0	0	X	1	0
Multichannel B Channels	0	0	X	0	1
Multichannel A and B Channels	0	0	X	1	1

Data Word Formats

The format of the data words transmitted over the serial ports is configured by the DTYPE, LSBF, SLEN, and PACK bits of the SPCTLx control registers.

Word Length (SLEN)

Serial ports can process word lengths of 3 to 32 bits for serial and multichannel modes and 8 to 32 bits for I^2S and left-justified modes. Word length is configured using the 5-bit SLEN field in the SPCTLx registers. Refer to Table 6-5 for further information.

The value of SLEN is: SLEN = serial word length – 1

Do not set the SLEN value to 0 or 1. Words smaller than 32 bits are right-justified in the receive and transmit buffers, residing in the least significant (LSB) bit positions (Table 6-6).

Table 6-6. Data Length versus Modes

Mode	Word Length (SLEN) bits
Standard Serial Mode	3–32
Left justified	8–32

Table 6-6. Data Length versus Modes (Cont'd)

Mode	Word Length (SLEN) bits
I^2S	8–32
Multichannel 3–32	

Although the word lengths can be 3 to 32 bits, transmitting or receiving words smaller than 7 bits at one-fourth the full peripheral clock rate of the serial port may cause incorrect operation when DMA chaining is enabled. Chaining locks the processor's internal I/O bus for several cycles while the new transfer control block (TCB) parameters are being loaded. Receive data may be lost (for example, overwritten) during this period. Moreover, transmitting or receiving words smaller than five bits may cause incorrect operation when all the DMA channels are enabled with no DMA chaining.

Endian Format (LSBF)

Endian format determines whether serial words transmit MSB first or LSB first. Endian format is selected by the LSBF bit in the SPCTLx registers. When LSBF = 0, serial words transmit (or receive) MSB first. When LSBF = 1, serial words transmit (or receive) LSB first.

Data Packing (PACK)

Received data words of 16 bits or less may be packed into 32-bit words, and 32-bit words being transmitted may be unpacked into 16-bit words. Word packing and unpacking is selected by the PACK bit in the SPCTLx control registers.

When PACK = 1 in the control register, two successive received words are packed into a single 32-bit word, and each 32-bit word is unpacked and transmitted as two 16-bit words.

The first 16-bit (or smaller) word is right-justified in bits 15–0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31–16. This applies to both receive (packing) and transmit (unpacking) operations. Companding can be used with word packing or unpacking.

When serial port data packing is enabled, the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

When 16-bit received data is packed into 32-bit words and stored in normal word space in processor's internal memory, the 16-bit words can be read or written with short word space addresses.

Data Type (DTYPE)

The DTYPE field of the SPCTLx registers specifies one of four data formats (for non-multichannel operation) shown in Table 6-4 on page 6-11. This bit field is reserved in I^2S and left-justified mode. In standard serial mode, if companding is selected for primary A channel, the secondary B channel performs a zero-fill.

In multichannel mode, channel B looks at XDTYPE[0] only.

If DTYPE[0] = 1 sign-extend
If DTYPE[0] = 0 zero-fill

These formats are applied to serial data words loaded into the receive and transmit buffers. Transmit data words are not zero-filled or sign-extended, because only the significant bits are transmitted.

Linear transfers occur in the primary channel, if the channel is active and companding is not selected for that channel (see "Companding the Data Stream" below). Companded transfers occur if the channel is active and companding is selected for that channel. The multichannel compand select registers, MTxCCSy and MRxCCSy, specify the transmit and receive channels that are companded when multichannel mode is enabled.

Transmit or receive sign extension is selected by bit 0 of DTYPE in the SPCTLx register and is common to all transmit or receive channels. If bit 0 of DTYPE is set, sign extension occurs on selected channels that do not have companding selected. If this bit is not set, the word contains zeros in the MSB positions. Companding is not supported for B channels. For B channels, transmit or receive sign extension is selected by bit 0 of DTYPE in the SPCTLx register.

Companding the Data Stream

Companding (compressing/expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits that must be sent. The processor's serial ports support the two most widely used companding algorithms, A-law and μ-law, performed according to the CCITT G.711 specification. The type of companding can be selected independently for each SPORT. Companding is selected by the DTYPE field of the SPCTLx control register.

Companding is supported on the A channel only. SPORT0, 2, and 4 primary channels are capable of compression, while SPORTs 1, 3 and 5 primary channels are capable of expansion.
In multichannel mode, when compression and expansion is enabled, the number of channels must be programmed via the NCH bit in the SPMCTLx registers before writing to the transmit FIFO. The SPxCSn and SPxCCsn registers should also be written before writing to transmit FIFO.

When companding is enabled, the data in the RXSPxA buffers is the right-justified, sign-extended expanded value of the eight received LSBs. A write to TXSPxA compresses the 32-bit value to eight LSBs (zero-filled to the width of the transmit word) before it is transmitted. If the 32-bit value is greater than the 13-bit A-law or 14-bit μ-law maximum, it is automatically compressed to the maximum value.

Companding As a Function

Since the values in the transmit and receive buffers are actually companded in place, the companding hardware can be used without transmitting (or receiving) any data, for example during testing or debugging. This operation requires one peripheral clock cycle of overhead, as described below. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

To compress data in place without transmitting, use the following procedure.

1. Set the SPTRAN bit to 1 in the SPCTLx register. The SPEN_A and SPEN_B bits should be = 0.
2. Enable companding in the DTYPE field of the SPCTLx transmit control register.
3. Write a 32-bit data word to the transmit buffer. Companding is calculated in this cycle.
4. Wait two cycles. Any instruction not accessing the transmit buffer can be used to cause this delay. This allows the serial port companding hardware to reload the transmit buffer with the companded value.
5. Read the 8-bit compressed value from the transmit buffer.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer. When expanding data in this way, set the appropriate serial word length (SLEN) in the SPCTLx register.

With companding enabled, interfacing the serial port to a codec requires little additional programming effort. If companding is not selected, two formats are available for received data words of fewer than 32 bits—one that fills unused MSBs with zeros, and another that sign-extends the MSB into the unused bits.

Clock Signal Options

Each serial port has a clock signal (SPORTx_CLK) for transmitting and receiving data on the two associated data signals. The clock signals are configured by the ICLK and CKRE bits of the SPCTLx control registers. A single clock signal clocks both A and B data signals (either configured as inputs or outputs) to receive or transmit data at the same rate.

Master Clock Divider Registers (DIVx)

The DIVx registers contain divisor values that determine frequencies for internally-generated clocks and frame syncs. The DIVx registers are described in "SPORT Divisor Registers (DIVx)" on page A-46.

If your system requires more precision and less noise and jitter, refer to "Precision Clock Generator" in Chapter 13, Precision Clock Generator.

Master Clock

The CLKDIV bit field specifies how many times the processor's internal clock (CCLK) is divided to generate the transmit and receive clocks. The frame sync (SPORTx_FS) is considered a receive frame sync if the data signals are configured as receivers. Likewise, the frame sync is considered a transmit frame sync if the data signals are configured as transmitters. The divisor is a 15-bit value, (bit 0 in divisor register is reserved) allowing a wide range of serial clock rates. Use the following equation tocalculate the serial clock frequency:

$$SCLK = \text{PCLK}/4(CLKDIV + 1)$$

The maximum serial clock frequency is equal to one-fourth (0.25) the processor's internal peripheral clock (PCLK) frequency, which occurs when CLKDIV is set to zero. Use the following equation to determine the value of CLKDIV, given the CCLK frequency and desired serial clock frequency:

CLKDIV = (PCLK/4 x SCLK) – 1

Master Frame Sync

The bit field FSDIV specifies how many transmit or receive clock cycles are counted before a frame sync pulse is generated. In this way, a frame sync can initiate periodic transfers. The counting of serial clock cycles applies to internally- or externally-generated serial clocks. The formula for the number of cycles between frame sync pulses is:

Number of serial clocks between frame syncs = FSDIV + 1

Use the following equation to determine the value of FSDIV, given the serial clock frequency and desired frame sync frequency:

FSDIV = (SCLK/4 x FSCLK) – 1

The frame sync is continuously active when FSDIV = 0. The value of FSDIV should not be less than the serial word length minus one (the value of the SLEN field in the serial port control register), as this may cause an external device to abort the current operation or cause other unpredictable results. If the serial port is not being used, the FSDIV divisor can be used as a counter for dividing an external clock or for generating a periodic pulse or periodic interrupt. The serial port must be enabled for this mode of operation to work properly.

Programs should not use master clock/frame sync on SPORTs to drive ADCs/DACs in high fidelity audio systems. Use the precision clock generator (PCG) instead.

Slave Mode

Exercise caution when operating with externally-generated transmit clocks near the frequency of PCLK/4 of the processor's internal clock. There is a delay between when the clock arrives at the SPORTx_CLK node and when

data is output. This delay may limit the receiver's speed of operation. Refer to the *ADSP-2136x SHARC Processor Data Sheet* for exact timing specifications.

Externally-generated late transmit frame syncs also experience a delay from when they arrive to when data is output. This can also limit the maximum serial clock speed. Refer to the *ADSP-2136x SHARC Processor Data Sheet* for exact timing specifications.

Clock Source (ICLK, MSTR)

The serial clock can be independently generated internally or input from an external source. The ICLK bit of the SPCTLx control registers determines the clock source.

When ICLK is set (=1), the clock signal is generated internally by the processor and the SPORTx_CLK signals are outputs. The clock frequency is determined by the value of the serial clock divisor (CLKDIV) in the DIVx registers.

When ICLK is cleared (=0), the clock signal is accepted as an input on the SPORTx_CLK signals, and the serial clock divisors in the DIVx registers are ignored.

The externally-generated serial clock does not need to be synchronous with the processor's system clock.

Sampling Edge (CKRE)

Data and frame syncs can be sampled on the rising or falling edges of the serial port clock signals. The CKRE bit of the SPCTLx control registers selects the sampling edge.

For sampling receive data and frame syncs, setting CKRE to 1 in the SPCTLx register selects the rising edge of SPORTx_CLK. When CKRE is cleared (=0), the processor selects the falling edge of SPORTx_CLK for sampling receive

data and frame syncs. Note that transmit data and frame sync signals change their state on the clock edge that is not selected.

For example, the transmit and receive functions of any two serial ports connected together should always select the same value for CKRE so internally-generated signals are driven on one edge and received signals are sampled on the opposite edge.

Frame Sync Options

Framing signals indicate the beginning of each serial word transfer. The SPORTx_FS signals are independent and are separately configured in the control registers. A variety of framing options are available on the SPORTs as shown in Table 6-7.

Table 6-7. Framing Options

OPMODE	Frame Sync Sampling
Standard serial	Level sensitive
Left-justified pair	Edge sensitive
I^2S	Edge sensitive
Multichannel	Level sensitive

Framed Versus Unframed Frame Syncs

The use of frame sync signals is optional in serial port communications. The FSR (transmit frame sync required) bit determines whether frame sync signals are required. Active low or active high frame syncs are selected using the LFS bit. This bit is located in the SPCTLx control registers.

When FSR is set (=1), a frame sync signal is required for every data word. To allow continuous transmission from the processor, each new data word

must be loaded into the transmit buffer before the previous word is shifted out and transmitted.

When FSR is cleared (=0), the corresponding frame sync signal is not required. A single frame sync is required to initiate communications but it is ignored after the first bit is transferred. Data words are then transferred continuously in what is referred to as an unframed mode.

When DMA is enabled in a mode where frame syncs are not required, DMA requests may be held off by chaining or may not be serviced frequently enough to guarantee continuous unframed data flow.

Figure 6-3 illustrates framed serial transfers.

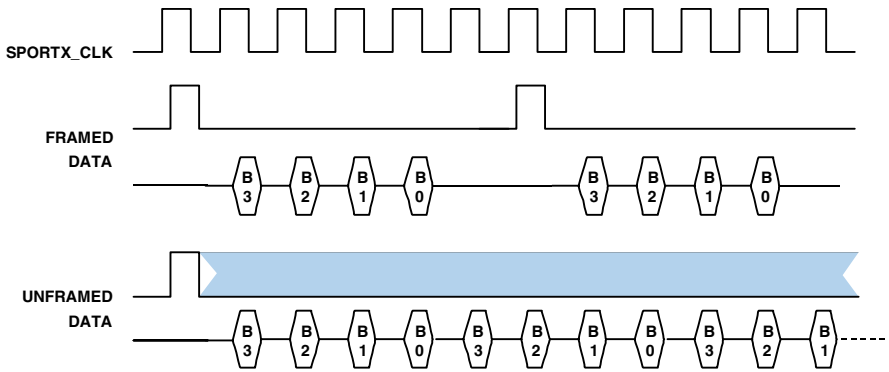


Figure 6-3. Framed Versus Unframed Data

Internal Versus External Frame Syncs (IFS, IMFS, MSTR)

Both transmit and receive frame syncs can be generated internally or input from an external source. The IFS bit of the SPCTLx control registers determines the frame sync source.

When IFS is set (=1), the corresponding frame sync signal is generated internally by the processor, and the SPORTx_FS signal is an output. The frequency of the frame sync signal is determined by the value of the frame sync divisor (FSDIV) in the DIVx registers.

When IFS is cleared (=0), the corresponding frame sync signal is accepted as an input on the SPORTx_FS signals, and the frame sync divisors in the DIVx registers are ignored.

Bit 14 (IMFS) in the SPCTLx registers selects whether the serial port uses an internally-generated frame sync (if set, =1) or frame sync from an external (if cleared, =0) source.

All frame sync options are available whether the signal is generated internally or externally.

Note that for I^2S and left-justified mode, the MSTR bit allows programs to select only the clock and word to be simultaneously configured as master or slave.

Logic Level Frame Syncs (LFS, LMFS)

Frame sync signals may be active high or active low (for example, inverted). The LFS bit (bit 16, or LMFS/LTDV bits for multichannel mode) of the SPCTLx control register determines the frame sync's logic level:

- When LFS is cleared (=0), the corresponding frame sync signal is active high.
- When LFS is set (=1), the corresponding frame sync signal is active low.

The LFS bit in the SPCTLx registers selects the logic level of the multichannel frame sync signals as active low (inverted) if set (=1) or active high if cleared (=0). Active high (=0) is the default.

Bit 16 in the SPCTLx registers selects the logic level of the transmit data valid signal (SPORTx_TDV_O) as active low (inverted) if set (=1), or active high if cleared (=0). These signals are actually SPORTx_FS, reconfigured as outputs during multichannel operation. They indicate which time slots have valid data to transmit.

Early Versus Late Frame Syncs (LAFS)

Frame sync signals can be early or late. Frame sync signals can occur during the first bit of each data word or during the serial clock cycle immediately preceding the first bit. The LAFS bit of the SPCTLx control register configures this option.

When LAFS is cleared (=0), early frame syncs are configured. This is the normal mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the serial clock cycle after the frame sync is asserted. The frame sync is not checked again until the entire word has been transmitted (or received). In multichannel operation, this is the case when the frame delay is one.

If data transmission is continuous in early framing mode (for example, the last bit of each word is immediately followed by the first bit of the next word), the frame sync signal occurs during the last bit of each word. Internally-generated frame syncs are asserted for one clock cycle in early framing mode.

When LAFS is set (=1), late frame syncs are configured. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the same serial clock cycle that the frame sync is asserted. In multichannel operation, this is the case when frame delay is zero. Receive data bits are latched by serial clock edges, but the frame sync signal is checked only during the first bit of each word. Internally-generated frame syncs remain asserted for the entire length of the data word in late framing mode. Externally-generated frame syncs are only

checked during the first bit. They do not need to be asserted after that time period.

Figure 6-4 illustrates the two modes of frame signal timing.

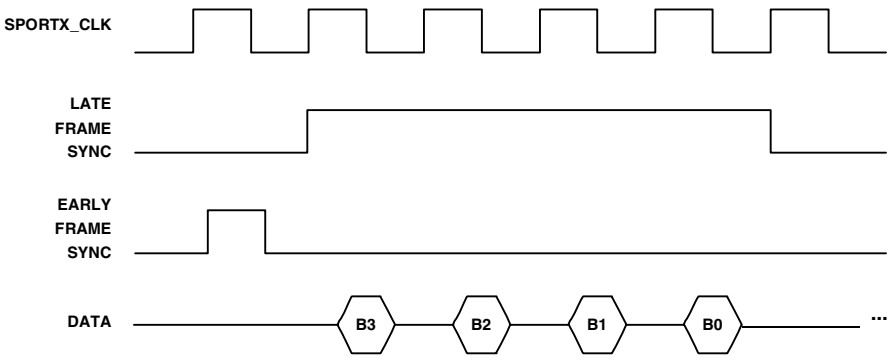


Figure 6-4. Normal Versus Alternate Framing

Data-Independent Frame Sync (One Channel)

When transmitting data out of the SPORT (SPTRAN = 1), the internally-generated frame sync signal normally is output-only when the transmit buffer has data ready to transmit. The data-independent frame sync mode allows the continuous generation of the SPORTx_FS signal, with or without new data in the register. The DIFS bit of the SPCTLx control registers configure this option.

When SPTRAN = 1, the DIFS bit selects whether the serial port uses a data-independent transmit frame sync (sync at selected interval, if set to 1) or a data-dependent transmit frame sync. When SPTRAN = 0, this bit selects whether the serial port uses a data-independent receive frame sync or a data-dependent receive frame sync.

When DIFS = 0 and SPTRAN = 1, the internally-generated transmit frame sync is only output when a new data word has been loaded into the

SPORT channel's transmit buffer. Once data is loaded into the transmit buffer, it is not transmitted until the next frame sync is generated. This mode of operation allows data to be transmitted only at specific times.

When DIFS = 0 and SPTRAN = 0, a receive SPORTx_FS signal is generated only when receive data buffer status is not full.

When DIFS = 1 and SPTRAN = 1, the internally-generated transmit frame sync is output at its programmed interval regardless of whether new data is available in the transmit buffer. The processor generates the transmit SPORTx_FS signal at the frequency specified by the value loaded in the DIV register. If a frame sync occurs when the transmitter FIFO is empty, the MSB or LSB (depending on how the LSBF bit in the SPCTLx registers is set) of the previous word is transmitted.

When DIFS = 1 and SPTRAN = 0, a receive SPORTx_FS signal is generated regardless of the receive data buffer status.

If the internally-generated frame sync is used and DIFS = 0, a single write to the transmit data register is required to start the transfer.

Data Independent Frame Sync (Two Channels)

When using both SPORT channels (SPORTx_DA and SPORTx_DB) as transmitters/receivers and MSTR = 1, SPTRAN = 0/1, and DIFS = 0, the processor generates a frame sync signal only when both transmit/receive buffers contain data because both transmitters/receivers share the same CLKDIV and SPORTx_FS.

For continuous transmission, both transmit buffers must contain new data.

When using both SPORT channels as transmitters and MSTR = 1, SPTRAN = 1 and DIFS = 1, the processor generates a frame sync signal at the frequency set by the FSDIVx bits, whether or not the transmit buffers contain new data.

Note that the SPORT DMA controller typically keeps the transmit buffer full. The application is responsible for filling the transmit buffers with data.

Operating Modes

The following sections provide detailed information on each operating mode available using the serial ports. It should be noted that many bits in the SPORT registers that control the function of the mode are the same bit but have a different name depending on the operating mode. Further, some bits are used in some modes but not others. For reference, see Table 6-4 on page 6-11, Table 6-5 on page 6-19, and "SPORT Serial Control Registers (SPCTLx)" on page A-30.

Standard Serial Mode

The standard serial mode lets programs configure serial ports for use by a variety of serial devices such as serial data converters and audio codecs. For more information, see Appendix C, Audio Frame Formats.

Timing Control Bits

Several bits in the SPCTLx control register enable and configure standard serial mode operation:

- Operation mode OPMODE = 0 (standard serial mode)
- Frame Sync Channel First (LFS)
- Frame Sync Required (FSR)
- Internal Frame Sync (IFS)
- Sampling Edges Frame Sync/data (CKRE)
- Logic Level Frame Sync (LFS)

- Internal clock enable (ICLK)
- Word length (SLEN, 3–32 bits)
- Channel enable (SPEN_A or SPEN_B)

Clocking Options

In standard serial mode, the serial portscan either accept an external serial clock or generate it internally. The ICLK bit in the SPCTL register determines the selection of these options. For internally-generated serial clocks, the CLKDIV bits in the DIVx register configure the serial clock rate.

Finally, programs can select whether the serial clock edge is used for sampling or driving serial data and/or frame syncs. This selection is performed using the CKRE bit in the SPCTL register. See "SPORT Serial Control Registers (SPCTLx)" on page A-30 for more details.

Frame Sync Options

A variety of framing options are available for the serial ports. In this mode, the options are independent of clocking, data formatting, or other configurations. The frame sync signal (SPORTx_FS) is used as a framing signal for serial word transfers.

Framing is optional for serial communications. The FSR bit in the SPCTL register controls whether the frame sync signal is required for every serial word transfer or if it isused simply to start a block of serial word transfers. Similar to the serial clock, the frame sync can be an external signal or generated internally. The IFS bit in the SPCTL register allows the selection between these options.

For internally-generated frame syncs, the FSDIV bits in the DIVx registers configure the frame sync rate. For internally-generated frame syncs, it is also possible to configure whether the frame sync signal is activated based on the FSDIV setting and the transmit or receive buffer status, or by the FSDIV setting only. All settings are configured through the DIFS bit of the

SPCTL register. The frame sync can be configured to be active high or active low through the LFS bit in the SPCTL register. The timing between the frame sync signal and the first bit of data either transmitted or received is also selectable through the LAFS bit in the SPCTL register.

Left-Justified Mode

Left-justified mode is a mode where in each frame sync cycle two samples of data are transmitted/received—one sample on the high segment of the frame sync, the other on the low segment of the frame sync. For more information, see Appendix C, Audio Frame Formats.

Programs have control over various attributes of this mode. One attribute is the number of bits (8-to 32-bit word lengths). However, each sample of the pair that occurs on each frame sync must be the same length. Set the late frame sync bit (LAFS bit) = 1 for left-justified mode.

Companding is not supported in left-justified or I^2S mode.

Each SPORT transmit or receive channel has a buffer enable, DMA enable, and chaining enable bits in its SPCTLx control register. The SPORTx_FS signal is used as the transmit and/or receive word select signal. DMA-driven or interrupt-driven data transfers can also be selected using bits in the SPCTLx register.

Master Serial Clock and Frame Sync Rates

The serial clock rate (CLKDIV value) for internal clocks can be set using a bit field in the DIVx register and the frame sync rate for internal frame sync can be set using the FSDIV bit field in the DIVx register. For details, see "SPORT Divisor Registers (DIVx)" on page A-46.

The transmitter sends the MSB of the next word in the same clock cycle as the word select (SPORTx_FS) signal changes.

To transmit or receive words continuously in left-justified mode, load the FSDIV register with the same value as SLEN. For example, for 8-bit data words (SLEN = 7), set FSDIV = 7.

Left-Justified Mode Timing Control Bits

Several bits in the SPCTLx control register enable and configure left-justified mode operation:

- Operation mode (OPMODE = 1, left-justified mode)
- Late frame sync (LAFS = 1, left-justified mode)
- Frame sync channel first (L_FIRST)
- Master mode enable (MSTR)
- Word length (SLEN, 8–32 bits)
- Channel enable (SPEN_A or SPEN_B)

For complete descriptions of these bits, see "SPORT Serial Control Registers (SPCTLx)" on page A-30.

Frame Sync Channel First (L_FIRST)

Using the L_FIRST bit, it is possible to select on which frame sync edge (rising or falling) that the SPORTs transmit or receive the first sample. Setting the L_FIRST bit to 1 = frame on falling edge of frame sync; 0 = frame on rising edge of frame sync.

Figure 6-5 illustrates only one possible combination of settings attainable in the left-justified mode. In this example case, OPMODE = 1, LAFS = 1, and L_FIRST = 1.

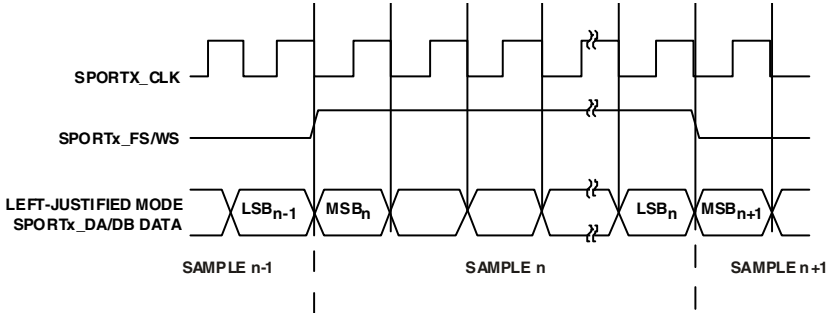


Figure 6-5. Word Select Timing in Left-justified Mode

I²S Mode

I²S mode is a three-wire serial bus standard protocol for transmission of two-channel (stereo) pulse code modulation (PCM) digital audio data. For more information, see Appendix C, Audio Frame Formats.

The I²S bus transmits audio data and control signals over separate lines. The data line carries two multiplexed data channels—the left channel and the right channel. In I²S mode, if both channels on a SPORT are set up to transmit, then SPORT transmit channels (TXSPxA and TXSPxB) transmit simultaneously, each transmitting left and right I²S channels. If both channels on a SPORT are set up to receive, the SPORT receive channels (RXSPxA and RXSPxB) receive simultaneously, each receiving left and right I²S channels. Data is transmitted in MSB-first format.

The SPORTs are designed so that in I²S master mode, SPORTx_FS (used as an edge sensitive signal to select between left and right channel) is held at the last driven logic level and does not transition, to provide an edge, after the final data word is driven out. Therefore, while transmitting a fixed number of words to an I²S receiver that expects an SPORTx_FS edge to receive the incoming data word, the SPORT should send a dummy word after

transmitting the fixed number of words. The transmission of this dummy word toggles SPORTx_FS, generating an edge. Transmission of the dummy word is not required when the I²S receiver is a serial port.

I²S Mode Timing Control Bits

Several bits in the SPCTLx register enable and configure I²S mode operation:

- Late frame sync (LAFS = 0, I²S mode)
- Operation mode (OPMODE = 1, I²S mode)
- Frame sync channel first (L_FIRST = 1, I²S mode)
- Master mode enable (MSTR)
- Word length (SLEN, 8–32 bits)
- Channel enable (SPEN_A or SPEN_B)

I²S mode is simply a subset of the left-justified mode which can be invoked by setting OPMODE = 1, LAFS = 0, and L_FIRST = 1. Note that in I²S mode, the data is delayed by one SCLK cycle

Selecting Transmit and Receive Channel Order (L_FIRST)

In master and slave modes, it is possible to configure the I²S channel to which each SPORT channel transmits or receives first. By default, the SPORT channels transmit and receive on the right I²S channel first. The left and right I²S channels are time-duplexed data channels.

To select the channel order, set the L_FIRST bit (= 1) to transmit or receive on the left channel first, or clear the L_FIRST bit (= 0) to transmit or receive on the right channel first.

For I²S operation (L_FIRST = 1), the transfer starts on the left channel first.

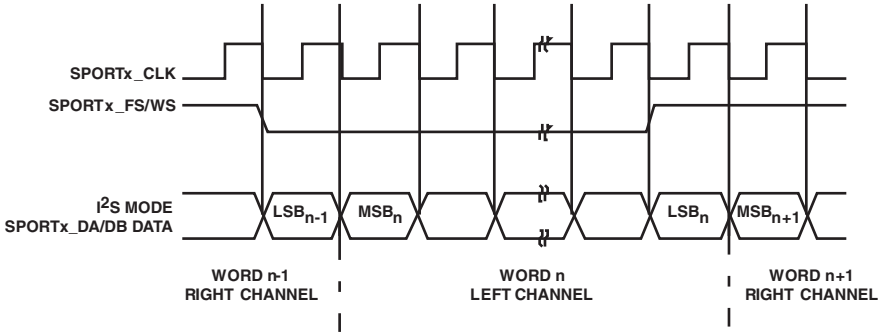


Figure 6-6. Word Select Timing in I²S Mode

Multichannel Operation

The processor's serial ports offer amultichannel mode of operation, which allows the SPORT to communicate ina time division multiplexed (TDM) serial system. For more information, see Appendix C, Audio Frame Formats.

The serial port can automatically select some words for particular channels while ignoring others. Up to 128 channels are available for transmitting or receiving or both. Each SPORT can receive or transmit data selectively from any of the 128 channels.

Data companding and DMA transfers can also be used in multichannel mode on channel A. Channel B can also be used in multichannel mode, but companding is not available on this channel.

Although the six SPORTs are programmable for data direction in the standard mode of operation, their programmability is restricted for multichannel operations. The following points summarize these limitations:

1. The primary A channels of SPORT1, 3, and 5 are capable of expansion only, and the primary A channels of SPORT0, 2, and 4 are capable of compression only.
2. In multichannel mode, SPORT0 and SPORT1 work in pairs; SPORT0 is the transmit channel, and SPORT1 is the receive channel. The same is true for SPORT2 and 3, and SPORT4 and 5.
3. Receive comparison is not supported.

SPORTs are forced into pairs in multichannel mode.

Table 6-8. TDM Signal Descriptions

Internal Nodes (routed to any DAI pin buffer)	Direction	Description
SPORT0/2/4_DA_O	O	**Data transmit channel A.** This signal is configured as an output to transmit serial data.
SPORT0/2/4_DB_O	O	**Data transmit channel B.** This signal is configured as an output to transmit serial data.
SPORT1/3/5_DA_I	I	**Data receive channel A.** This signal is configured as an input to receive serial data.
SPORT1/3/5_DB_I	I	**Data receive channel B.** This signal is configured as an input to receive serial data.
SPORT0/2/4_CLK_I	I	**Serial Input clock.** This signal can be either internally or externally generated.
SPORT1/3/5_CLK		**Not connected in TDM mode.** Clock signal provided internally by the SPORTx_CLK0/2/4 signal.
SPORT1/3/5_FS_I/O	I/O	**Multichannel frame sync.** SPORT 1 generates the frame sync pulse for the SPORT0/1 pair. SPORT 3 generates the frame sync pulse for the SPORT2/3 pair. SPORT 5 generates the frame sync pulse for the SPORT4/5 pair.

Table 6-8. TDM Signal Descriptions (Cont'd)

Internal Nodes (routed to any DAI pin buffer)	Direction	Description
SPORT01/23/45_TDV_O	O	**Transmit d ata valid.** SPORT 0 generates the transmit data valid pulse (TDV01) for the SPORT0/1 pair. SPORT 2 generates the transmit valid pulse (TDV23) for the SPORT2/3 pair. SPORT 4 generates the transmit valid pulse (TDV45) for the SPORT4/5 pair.
SPORT5–0_DA_PBEN_O	O	Only driven in master mode
SPORT5–0_DB_PBEN_O	O	
SPORT5–0_CLK_PBEN_O	O	
SPORT5–0_FS_PBEN_O	O	

Frame Syncs Signals

All receiving and transmitting devices in a multichannel system must have the same timing reference. The SPORT135_FS signal is used for this reference, indicating the start of a block(or frame) of multichannel data words. Pairs of SPORTs share thesame frame sync signalfor multichannel mode:

- SPORT1_FS for SPORT0/1
- SPORT3_FS for SPORT2/3
- SPORT5_FS for SPORT4/5

When multichannel mode is enabled on a SPORT0/1, SPORT2/3, or SPORT4/5 pair, both the transmitter and receiver use the SPORT1_FS, SPORT3_FS, or the SPORT5_FS signals respectively as a frame sync. This is true whether SPORT1_FS, SPORT3_FS, or the SPORT5_FS is generated internally or externally. This signal synchronizes the channels and restarts each multichannel sequence. The SPORT1_FS, SPORT3_FS, or SPORT5_FS signal initiates the beginning of the channel 0 data word.

SPORTs are paired when multichannel mode is selected. In this mode, transmit/receive directions are fixed where SPORTS 0, 2, and 4 act as transmitters, and SPORTs 1, 3, and 5 act as receivers.

Transmit Valid Signals

The SPORT0_FS, SPORT2_FS or SPORT4_FS are used as a transmit data valid signals, which are active during transmission of an enabled word. Because the serial port's SPORT0_DA/B, SPORT2_DA/B and SPORT4_DA/B signals are three-stated when the time slot is not active, the SPORT0_FS, SPORT2_FS, SPORT4_FS signal specifies if SPORT0_DA/B, SPORT2_DA/B, SPORT4_DA/B is being driven by the processor.

In multichannel mode, the SPORT0_FS signal is renamed TDV01, the SPORT2_FS signal is renamed TDV23 and the SPORT4_FS signal is renamed TDV45. These signals become outputs in this mode.

Do not connect SPORT0_FS (TDV01) to SPORT1_FS, and SPORT4_FS (TDV45) to SPORT5_FS in multichannel mode because bus contention between the transmit data valid and multichannel frame sync signals results.

After the TXSPxA transmit buffer is loaded, transmission begins and the SPORT0_FS, SPORT2_FS/SPORT4_FS signals are generated. When serial port DMA is used, this may occur several cycles after the multichannel transmission is enabled. If a deterministic start time is required, pre-load the transmit buffer.

Figure 6-7 shows an example of timing for a multichannel transfer with SPORT pairing. The transfer has the following characteristics:

- The transfer uses the TDM method where serial data is sent or received on different channels while sharing the same serial data bus.
- The SPORT1_FS signals the start of a frame for SPORT01 multichannel pairing.

- The TDV01(SPORT0_FS) is used as transmit data valid for external logic. This signal is active only during transmit channels.
- The transfer is received on channel 0 (word 0), and transmits on channels 1 and 2 (word 1 and 2)

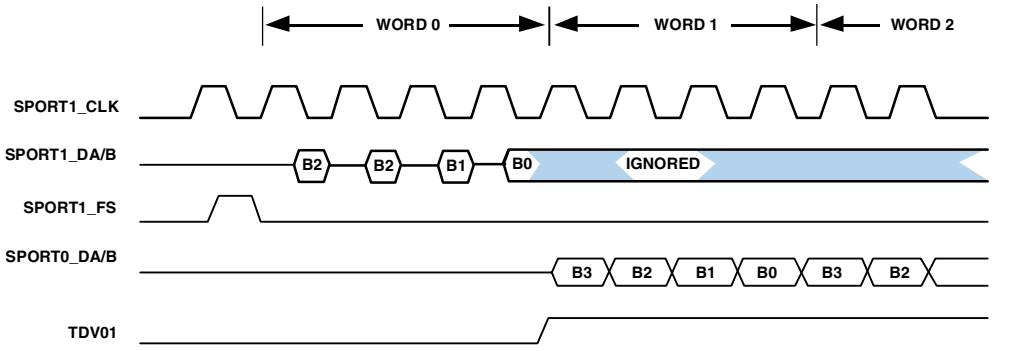


Figure 6-7. Multichannel Operation

Multichannel Mode Control Bits

Several bits in the SPCTLx control register enable and configure multichannel mode operation:

- Operation mode (OPMODE) = 0 (disable non-multichannel)
- Master mode (ICLK)
- Sampling edge frame sync/data (CKRE)
- Internal frame sync (IMFS)
- Logic level frame sync (LMFS/LTDV)
- Word length (SLEN) (3–32 bits)
- Channel enable (SPEN_A or SPEN_B)

Setting the MCEA or MCEB bits enables multichannel operation for both receive and transmit sides of the SPORT0/1, SPORT2/3 or SPORT4/5 pair.

Number of Channels (NCH)

Select the number of channels used in multichannel operation by using the 7-bit NCH field in the multichannel control register. Set NCH to the actual number of channels minus one: NCH = Number of channels – 1

The 7-bit CHNL field in the multichannel control registers indicates the channel that is currently selected during multichannel operation. This field is a read-only status indicator. TheCHNL(6:0) bits increment modulo NCH(6:0) as each channel is serviced.

Frame Delay (MFD)

The 4-bit MFD field (bits 4–1) in the multichannel control registers (SPMCTL01, SPMCTL23, and SPMCTL45) specifies a delay between the frame sync pulse and the first data bit in multichannel mode. The value of MFD is the number of serial clock cycles of the delay. Multichannel frame delay allows the processor to work with different types of telephony interface devices.

A value of zero for MFD causes the frame sync to be concurrent with the first data bit. The maximum value allowed for MFD is 15. A new frame sync may occur before data from the last frame has been received, because blocks of data occur back to back.

Channel Selection Registers

Specific channels can be individually enabled or disabled to select the words that are received and transmitted during multichannel communications. Data words from the enabled channels are received or transmitted,

while disabled channel words are ignored. Up to 128 channels are available for transmitting and receiving.

The multichannel selection registers enable and disable individual channels. The registers for each serial port are shown in "SPORT Transmit Select Registers (MTxCSy)" on page A-45, "SPORT Transmit Compand Registers (MTxCCSy)" on page A-45, "SPORT Receive Select Registers (MRxCSx)" on page A-46, and "SPORT Receive Compand Registers (MRxCCSx)" on page A-46.

Each of the four multichannel enable and compand select registers are 32 bits in length. These registers provide channel selection for 128 (32 bits x 4 channels = 128) channels. Setting a bit enables that channel so that the serial port selects its word from the multiple-word block of data (for either receive or transmit). For example, setting bit 0 in `MT0CS0` or `MT2CS0` selects word 0, setting bit 12 selects word 12, and so on. Setting bit 0 in `MT0CS1` or `MT2CS1` selects word 32, setting bit 12 selects word 44, and so on.

Companding Selection

Companding may be selected on a per-channel basis. Setting a bit to 1 in any of the multichannel registers specifies that the data be companded for that channel. A-law or μ-law companding can be selected using the `DTYPE` bit in the `SPCTLx` control registers. SPORT1, 3, and 5 expand selected incoming time slot data, while SPORT0, 2, and 4 compress selected outgoing time slot data.

Transmit Selection Registers

Setting a particular bit to 1 in the `MT0CS0-3`, `MT2CS0-3` or `MT4CS0-3` registers causes SPORT0, 2, or 4 to transmit the word in that channel's position of the data stream. Clearing the bit in the register causes the SPORT0 `SPORT0_DA/B`, SPORT2 `SPORT2_DA/B` or SPORT4's `SPORT4_DA` data transmit signal to three-state during the time slot of that channel.

Receive Selection Registers

Setting a particular bit to 1 in the MR1CS0-3, MR3CS0-3 or MR5CS0-3 register causes the serial port to receive the word in that channel's position of the data stream. The received word is loaded into the receive buffer. Clearing the bit in the register causes the serial port to ignore the data.

Data Transfer Types

Serial port data can be transferred for use by the processor in two different methods:

- Core-driven single word transfers
- DMA transfers between both internal and external memory

DMA transfers can be set up to transfer a configurable number of serial words between the serial port buffers (TXSPxA, TXSPxB, RXSPxA, and RXSPxB) and internal memory automatically. Core-driven transfers use SPORT interrupts to signal the processor core to perform single word transfers to/from the serial port buffers (TXSPxA, TXSPxB, RXSPxA, and RXSPxB).

Core Transfers

The following sections provide information on core driven data transfers.

Single Word Transfers

Individual data words may also be transmitted and received by the serial ports, with interrupts occurring as each 32-bit word is transmitted or received. When a serial port is enabled and DMA is disabled, the SPORT interrupts are generated whenever a complete 32-bit word has been received in the receive buffer, or whenever the transmit buffer is not full.

When performing core-driven transfers, write to the buffer designated by the SPTRAN bit setting in the SPCTLx registers. For DMA-driven transfers, the serial port logic performs the data transfer from internal memory to/from the appropriate buffer depending on the SPTRAN bit setting. If the inactive SPORT data buffers are read or written to by core while the port is being enabled, the core hangs. For example, if a SPORT is programmed to be a transmitter, while at the same time the core reads from the receive buffer of the same SPORT, the core hangs just as it would if it were reading an empty buffer that is currently active. This locks up the core until the SPORT is reset.

To avoid hanging the processor core, check the buffer's full/empty status when the processor core's program reads a word from a serial port's receive buffer or writes a word to its transmit buffer. This condition can also happen to an external device, for example a host processor, when it is reading or writing a serial port buffer. The full/empty status can be read in the DXS bits of the SPCTLx register. Reading from an empty receive buffer or writing to a full transmit buffer causes the processor (or external device) to hang, while it waits for the status to change.

Internal Memory DMA Transfers

Transmit and receive data can be transferred between each serial port and both on-chip and off-chip memory with single word transfers or with DMA block transfers. Unlike the previous processors, ADSP-2136x processors allow data transfer directly between SPORTs and external memory using DMA in addition to transfers to internal memory. Note that since data transfer from SPORT to external memory is possible only through DMA, any reference in this chapter to non-DMA transfer methods refer to transfers to on-chip memory. Both methods are interrupt-driven, and use the same internally-generated interrupts.

SPORT DMA provides a mechanism for receiving or transmitting an entire block of serial data before the interrupt is generated. When serial port DMA is not enabled, the SPORT generates an interrupt every time it

receives or starts to transmit a data word. The processor's on-chip DMA controller handles the DMA transfer, allowing the processor core to continue running until the entire block of data is transmitted or received. Service routines can then operate on the block of data rather than on single words, significantly reducing overhead.

Therefore, set the direction bit, the serial port enable bit, and DMA Enable bits before initiating any operations on the SPORT data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, it can cause unpredictable results.

Each transmitter and receiver has its own DMA registers. The same DMA channel drives the left and right I^2S channels for the transmitter or the receiver. The software application must stop multiplexing the left and right channel data received by the receive buffer, because the left and right data are interleaved in the DMA buffers.

Channel A and B on each SPORT share a common interrupt vector. The DMA controller generates an interrupt at the end of DMA transfer only.

Figure 6-4 on page 6-32 shows the relationship between frame sync (word select), serial clock, and I^2S data. Timing for word select is the same as for frame sync.

The value of the SPTRAN bit in SPCTLx (0 = RX, 1 = TX) determines whether the receive or transmit register for the SPORT becomes active.

The SPORT DMA channels are assigned higher priority than all other DMA channels (for example, the SPI port) because of their relatively low service rate and their inability to hold off incoming data. Having higher priority causes the SPORT DMA transfers to be performed first when multiple DMA requests occur in the same cycle. The serial port DMA channels are numbered and prioritized as shown in Table 2-9 on page 2-11.

Although the DMA transfers are performed with 32-bit words, serial ports can handle word sizes from 3 to 32 bits, with 8 to 32 bits for I^2S mode. If serial words are 16 bits or smaller, they can be packed into 32-bit words for each DMA transfer. DMA transfers are configured using the PACK bit in the SPCTLx registers. When serial port data packing is enabled (PACK = 1), the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

Standard DMA

Each SPORT DMA channel has an enable bit (SDEN_A and SDEN_B) in its SPCTLx register. When DMA is disabled for a particular channel, the SPORT generates an interrupt every time it receives a data word or whenever there is a vacancy in the transmit buffer. For more information, see "Single Word Transfers" on page 6-47.

To set up a serial port DMA channel, write a set of memory buffer parameters to the SPORT DMA parameter registers as shown in Table 2-9 on page 2-11.

Load the II, IM, and C registers with a starting address for the buffer, an address modifier, and a word count, respectively. The index register contains the internal memory address for transfers to internal memory and the external memory address for transfers to external memory. These registers can be written from the core processor or from an external processor.

Once serial port DMA is enabled, the processor's DMA controller automatically transfers received data words in the receive buffer to the buffer in internal or external memory, depending on the transfer type. Likewise, when the serial port is ready to transmit data, the DMA controller automatically transfers a word from internal or external memory to the transmit buffer. The controller continues these transfers until the entire data buffer is received or transmitted.

When the count register of an active DMA channel reaches zero (0), the SPORT generates the corresponding interrupt.

DMA Chaining

Each channel also has a DMA chaining enable bit (SCHEN_A and SCHEN_B) in its SPCTLx control register.

Each SPORT DMA channel also has a chain pointer register (CPSPxy). The CPSPxy register functions are used in chained DMA operations.

In chained DMA operations, the processor's DMA controller automatically sets up another DMA transfer when the contents of the current buffer have been transmitted (or received). The chain pointer register (CPSPxy) functions as a pointer to the next set of buffer parameters stored in external or internal memory. The DMA controller automatically downloads these buffer parameters to set up the next DMA sequence. For more information on SPORT DMA chaining, see "DMA Chaining" on page 2-20.

DMA chaining occurs independently forthe transmit and receive channels of each serial port. Each SPORT DMA channel has a chaining enable bit (SCHEN_A or SCHEN_B) that when set (= 1), enables DMA chaining and when cleared (= 0), disables DMA chaining. Writing all zeros to the address field of the chain pointer register (CPSPxy) also disables chaining.

The chain pointer register should be cleared first before chaining is enabled.

The I/O processor responds by auto-initializing the first DMA parameter registers with the values from the first TCB, and then starts the first data transfer.

DMA Chain Insertion Mode

It is possible to insert a single SPORT DMA operation or another DMA chain within an active SPORT DMA chain. Programs may need to perform insertion when a high priority DMA requires service and cannot wait for the current chain to finish.

When DMA on a channel is disabled and chaining on the channel is enabled, the DMA channel is in chain insertion mode. This mode lets a program insert a new DMA or DMA chain within the current chain without effecting the current DMA transfer.

Chain insertion mode operates the same as non-chained DMA mode. When the current DMA transfer ends, an interrupt request occurs and no TCBs are loaded. This interrupt request is independent of the `PCI` bit state.

Chain insertion should not be set up as an initial mode of operation. This mode should only be used to insert one or more TCBs into an active DMA chaining sequence. For more information, see "Enter DMA Chain Insertion Mode" on page 6-57.

Interrupts

This section handles the various scenarios in which an interrupt is triggered.

Internal Transfer Completion

Each serial port has an interrupt associated with it. For each SPORT, both the A and B channel transmit and receive data buffers share the same interrupt vector. The interrupts can be used to indicate the completion of the transfer of a block of serial data when the serial ports are configured for DMA. They can also be used to perform single word transfers (refer to

"Single Word Transfers" on page 6-47). The priority of the serial port interrupts is shown in Table 2-9 on page 2-11.

Multiple interrupts can occur if both SPORTs transmit or receive data in the same cycle. Any interrupt can be masked in the IMASK register; if the interrupt is later enabled in the LIRPTL register, the corresponding interrupt latch bit in the IRPTL or LIRPTL registers must be cleared in case the interrupt has occurred in the same time period.

SPORT interrupts occur on the second peripheral clock (PCLK) after the last bit of the serial word is latched in or driven out.

When serial port data packing is enabled (PACK = 1 in the SPCTLx registers), the transmit and receive interrupts are generated for 32-bit packed words, not for each 16-bit word.

Each DMA channel has a count register (CSPxA/CSPxB), which must be initialized with a word count that specifies the number of words to transfer. The count register decrements after each DMA transfer on the channel. When the word count reaches zero, the SPORT generates an interrupt, then automatically stops the DMA channel.

Shared Channels

Both the A and B channels share a common interrupt vector in the interrupt-driven data transfer mode, regardless of whether they are configured as a transmitter or receiver.

The SPORT generates an interrupt when the transmit buffer has a vacancy or the receive buffer has data. To determine the source of an interrupt, applications must check the transmit or receive data buffer status bits (DXS_A, DXS_B).

Debug Features

The following sections provide information on debugging features available with the serial ports.

SPORT Loopback

When the SPORT loopback bit, SPL (bit 12), is set in the SPMCTLxy register, the serial port is configured in an internal loopback connection as follows: SPORT0 and SPORT1 work as a pair for internal loopback, SPORT2 and SPORT3 workas pairs, and SPORT4 and SPORT5 work as pairs. The loopback mode enables programs to test the serial ports internally and to debug applications.

The SPL bit applies to standard serial and I^2S modes only.

When loopback is configured, the

- SPORTx_DA, SPORTx_DB, SPORTx_CLK, and SPORTx_FS signals of SPORT0 and SPORT1 are internally connected (where x = 0 or 1).
- The SPORTy_DA, SPORTy_DB, SPORTy_CLK, and SPORTy_FS signals of SPORT2 and SPORT3 are internally connected (where y = 2 or 3).
- The SPORTz_DA, SPORTz_DB, SPORTz_CLK and SPORTz_FS signals of SPORT4 and SPORT5 are internally connected (where z = 4 or 5).

In loopback mode, either of the two paired SPORTS can be transmitters or receivers. One SPORT in the loopback pair must be configured as a transmitter; the other must be configured as a receiver. For example, SPORT0 can be a transmitter and SPORT1 can be a receiver for internal loopback. Or, SPORT0 can be a receiver and SPORT1 can be the transmitter when setting up internal loopback. The processor ignores external activity on the SPORTx_CLK, SPORTx_FS A and B channel data signals when

the SPORT is configured in Loopback mode. This prevents contention with the internal loopback data transfer.

Only transmit clock and transmit frame sync options may be used in loopback mode—programs must ensure that the serial port is set up correctly in the SPCTLx control registers. Multichannel mode is not allowed. Only standard serial, left-justified, and I^2S modes support internal loopback. In loopback, each SPORT can be configured as transmitter or receiver, and each one is capable of generating an internal frame sync and clock.

Any of the four paired SPORTs can be set up to transmit or receive, depending on their SPTRAN bit configurations

Loopback Routing

The SPORTs support an internal loopback mode by using the SRU. For more information, see “Loop Back Routing” on page 5-30.

Buffer Hang Disable (BHD)

To support debugging buffer transfers, the ADSP-2136x processors have a buffer hang disable (BHD) bit. When set (= 1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition. For more information, see the BHD bit description on on page 6-12.

Effect Latency

SPORT control register writes are internally completed at the end of five core clock cycles. The newly written value to the SPORT control register can be read back on the next cycle. After a write to a SPORT control register, control and mode bit changes take effect in the second serial clock cycle (SCLK).

The SPORT is ready to start transmitting or receiving three serial clock cycles after they are enabled in the SPCTLx control register. No serial clocks are lost from this point on. This delay does also apply in slave mode (external clock/frame sync) for synchronization.

Multichannel operation is activated 3 serial clock cycles (SCLK) after the MCEA or MCEB bits are set. Internally-generated frame sync signals activate 4 serial clock cycles after the MCEA or MCEB bits are set.

For access latency of IOP registers see "I/O Processor Register Access" on page 2-32.

Programming Model

The section describes some programming procedures that are used to enable and operate the SPORTs.

Setting Up and Starting Chained DMA

To set up and initiate a chain of DMA operations, use the following procedure.

1. Clear the chain pointer register.
2. For internal memory transfers, set up all TCBs in internal memory. For external memory transfers, TCBs can be set up in internal or external memory.
3. Write to the appropriate DMA control register, setting the DMA enable bit to one and the chaining enable bit to one.
4. Write the address containing the index register value of the first TCB to the chain pointer register, which starts the chain.

Enter DMA Chain Insertion Mode

Chain insertion lets the SPORTs insert a single SPORT DMA operation or another DMA chain within an active SPORT DMA chain.

1. Enter chain insertion mode by setting SDENx = 0 and SCHENx = 1 in the channel's DMA control register. The DMA interrupt indicates when the current DMA sequence is complete.
2. Copy the address currently held in the chain pointer register to the chain pointer position of the last TCB in the chain that is being inserted.
3. Write the start address of the first TCB of the new chain into the chain pointer register.
4. Resume chained DMA mode by setting SDENx = 1 and SCHENx = 1.

Programming Examples

This section provides three programming examples written for the ADSP-2136x processor. The first listing, Listing 6-1, transmits a buffer of data from SPORT5 to SPORT4 using DMA and the internal loopback feature of the serial port. In this example, SPORT5 drives the clock and frame sync, and the buffer is transferred only one time.

The second listing, Listing 6-2, transmits a buffer of data from SPORT2 to SPORT3 using direct core reads and writes and the internal loopback feature of the serial port. In this example, SPORT2 drives the clock and frame sync, and the buffer is transferred only one time.

The third listing, Listing 6-3, transmits a buffer of data from SPORT1 to SPORT0 using DMA chaining and the internal loopback feature of the serial port. In this example, SPORT5 drives the clock and frame sync, and the two TCBs for each SPORT are set up to ping-pong back and forth to continually send and receive data.

Listing 6-1. SPORT Transmit Using DMA

```
#include <def21364.h>
#include <sru21364.h>
#include <SRU.h>

/* Default Buffer Length */
#define BUFSIZE 10
.SECTION/DM seg_dmda;
/* TX Buffers */
.var tx_buf1a[BUFSIZE] = 0x11111111,
0x22222222,
0x33333333,
0x44444444,
0x55555555,
0x66666666,
0x77777777,
0x88888888,
0x99999999,
0xAAAAAAAA;
.var tx_buf1b[BUFSIZE] = 0x12345678,
0x23456789,
0x3456789A,
0x456789AB,
0x56789ABC,
0x6789ABCD,
0x789ABCDE,
0x89ABCDEF,
0x9ABCDEF0,
0xABCDEF01;

/* RX Buffers */
.var rx_buf0a[BUFSIZE];
.var rx_buf0b[BUFSIZE];
```

```
/* TX Transfer Control Blocks */
.var tx_tcb1[4] = 0,BUFSIZE,1,tx_buf1a;
.var tx_tcb2[4] = 0,BUFSIZE,1,tx_buf1b;
/* RX Transfer Control Blocks */
.var rx_tcb1[4] = 0,BUFSIZE,1,rx_buf0a;
.var rx_tcb2[4] = 0,BUFSIZE,1,rx_buf0b;

/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:
r0 = 0; /* Clear CP registers before enabling chaining */
dm(CPSP0A) = r0;
dm(CPSP1A) = r0;
/* SPORT Loopback: Use SPORT0 as RX & SPORT1 as TX */
/* initially clear SPORT control register */
r0 = 0x00000000;
dm(SPCTL0) = r0;
dm(SPCTL1) = r0;
dm(SPMCTL01) = r0;
SPORT_DMA_setup:
/* set internal loopback bit for SPORT0 & SPORT1 */
bit set ustat3 SPL;
dm(SPMCTL01) = ustat3;

/* Configure SPORT1 as a transmitter */
/* internally generating clock and frame sync */
/* CLKDIV = [fPCLK(167 MHz)/4xFSCLK(8.325 MHz)]-1 = 0x0004 */
/* FSDIV = [FSCLK(8.325 MHz)/TFS(260 kHz)]-1 = 31 = 0x001F */
R0 = 0x001F0004; dm(DIV1) = R0;
ustat4 = SPEN_A| /* Enable Channel A */
SLEN32| /* 32-bit word length */
FSR| /* Frame Sync Required */
SPTRAN| /* Transmit on enabled channels */
```

```
SDEN_A| /* Enable Channel A DMA */
SCHEN_A| /* Enable Channel A DMA Chaining */
IFS| /* Internally-generated Frame Sync */
ICLK; /* Internally-generated Clock */
dm(SPCTL1) = ustat4;

/* Configure SPORT0 as a receiver */
/* externally generating clock and frame sync */
r0 = 0x0; dm(DIV0) = R0;
ustat3 = SPEN_A| /* Enable Channel A */
SLEN32| /* 32-bit word length */
FSR| /* Frame Sync Required */
SDEN_A| /* Enable Channel A DMA */
SCHEN_A; /* Enable Channel A DMA Chaining */
dm(SPCTL0) = ustat3;

/* Next TCB location for tx_tcb2 is tx_tcb1 */
/* Mask the first 19 bits of the TCB location */
r0 = (tx_tcb1 + 3) & 0x7FFFF;
dm(tx_tcb2) = r0;

/* Next TCB location for rx_tcb2 is rx_tcb1 */
/* Mask the first 19 bits of the TCB location */
r0 = (rx_tcb1 + 3) & 0x7FFFF;
dm(rx_tcb2) = r0;

/* Next TCB location for rx_tcb1 is rx_tcb2 */
/* Mask the first 19 bits of the TCB location */
r0 = (rx_tcb2 + 3) & 0x7FFFF;
dm(rx_tcb1) = r0;

/* Initialize SPORT DMA transfer by writing to the CP reg */
dm(CPSP0A) = r0;
```

```
/* Next TCB location for tx_tcb1 is tx_tcb2 */
/* Mask the first 19 bits of the TCB location */
r0 = (tx_tcb2 + 3) & 0x7FFFF;
dm(tx_tcb1) = r0;

/* Initialize SPORT DMA transfer by writing to the CP reg */
dm(CPSP1A) = r0;
_main.end: jump (pc,0);
```

Listing 6-2. SPORT Transmit Using Direct Core Access

```
/* Default Buffer Length */
#define BUFSIZE 10
.SECTION/DM seg_dmda;

/* Transmit Buffer */
.var tx_buf2a[BUFSIZE] = 0x11111111,
0x22222222,
0x33333333,
0x44444444,
0x55555555,
0x66666666,
0x77777777,
0x88888888,
0x99999999,
0xAAAAAAAA;

/* Receive Buffer */
.var rx_buf3a[BUFSIZE];

/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:
```

```
bit set mode1 CBUFEN;
/* enable circular buffers */
/* SPORT Loopback: Use SPORT2 as RX & SPORT3 as TX */
/* Initially clear SPORT control registers */
r0 = 0x00000000;
dm(SPCTL2) = r0;
dm(SPCTL3) = r0;
dm(SPMCTL23) = r0;


/* Set up DAG registers */
i4 = tx_buf2a;
m4 = 1;
l4 = 0;
i12 = rx_buf3a;
m12 = 1;
l12 = 0;
SPORT_DMA_setup:

/* set internal loopback bit for SPORT2 & SPORT3 */
bit set ustat3 SPL;
dm(SPMCTL23) = ustat3;

/* Configure SPORT2 as a transmitter */
/* internally generating clock and frame sync */
/* CLKDIV = [fPCLK(167 MHz)/4xFSCLK(8.325 MHz)]-1 = 0x0004 */
/* FSDIV = [FSCLK(8.325 MHz)/TFS(260 kHz)]-1 = 31 = 0x001F */
R0 = 0x001F0004; dm(DIV2) = R0;
ustat4 = SPEN_A|            /* Enable Channel A */
SLEN32|                     /* 32-bit word length */
FSR|                        /* Frame Sync Required */
SPTRAN|                     /* Transmit on enabled channels */
IFS|                        /* Internally Generated Frame Sync */
ICLK;                       /* Internally Generated Clock */
```

```
dm(SPCTL2) = ustat4;

/* Configure SPORT3 as a receiver */
/* externally generating clock and frame sync */
r0 = 0x0; dm(DIV3) = R0;
ustat3 = SPEN_A|            /* Enable Channel A */
SLEN32|                     /* 32-bit word length */
FSR;                        /* Frame Sync Required */
dm(SPCTL3) = ustat3;
/* Set up loop to transmit and receive data */
lcntr = LENGTH(tx_buf2a), do (pc,4) until lce;

/* Retrieve data using DAG1 and send TX via SPORT2 */
r0 = dm(i4,m4);
dm(TXSP2A) = r0;

/* Receive data via SPORT3 and save via DAG2 */
r0 = dm(RXSP3A);
pm(i12,m12) = r0;
_main.end: jump (pc,0);
```

Listing 6-3. SPORT Transmit Using DMA Chaining

```
/* Default Buffer Length */
#define BUFSIZE 10
.SECTION/DM seg_dmda;

/*Transmit buffer*/
.var tx_buf5a[BUFSIZE] = 0x11111111,
0x22222222,
0x33333333,
0x44444444,
0x55555555,
0x66666666,
```

```
0x77777777,
0x88888888,
0x99999999,
0xAAAAAAAA;
/*Receive buffer*/
.var rx_buf4a[BUFSIZE];



/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:

/* SPORT Loopback: Use SPORT4 as RX & SPORT5 as TX */
/* initially clear SPORT control register */
r0 = 0x00000000;
dm(SPCTL4) = r0;
dm(SPCTL5) = r0;
dm(SPMCTL45) = r0;
SPORT_DMA_setup:

/* SPORT 5 Internal DMA memory address */
r0 = tx_buf5a; dm(IISP5A) = r0;

/* SPORT 5 Internal DMA memory access modifier */
r0 = 1; dm(IMSP5A) = r0;

/* SPORT 5 Number of DMA transfers to be done */
r0 = length(tx_buf5a); dm(CSP5A) = r0;

/* SPORT 4 Internal DMA memory address */
r0 = rx_buf4a; dm(IISP4A) = r0;
```

```
/* SPORT 4 Internal DMA memory access modifier */
r0 = 1; dm(IMSP4A) = r0;

/* SPORT 4 Number of DMA5 transfers to be done */
r0 = length(rx_buf4a); dm(CSP4A) = r0;

/* set internal loopback bit for SPORT4 & SPORT5 */
bit set ustat3 SPL;
dm(SPMCTL45) = ustat3;

/* Configure SPORT5 as a transmitter */
/* internally generating clock and frame sync */
/* CLKDIV = [fPCLK(167 MHz)/4xFSCLK(8.325 MHz)]-1 = 0x0004 */
/* FSDIV = [FSCLK(8.325 MHz)/TFS(260 kHz)]-1 = 31 = 0x001F */
R0 = 0x001F0004; dm(DIV5) = R0;
ustat4 = SPEN_A|         /* Enable Channel A */
SLEN32|                  /* 32-bit word length */
FSR|                     /* Frame Sync Required */
SPTRAN|                  /* Transmit on enabled channels */
SDEN_A|                  /* Enable Channel A DMA */
IFS|                     /* Internally Generated Frame Sync */
ICLK;                    /* Internally Generated Clock */
dm(SPCTL5) = ustat4;

/* Configure SPORT4 as a receiver */
/* externally generating clock and frame sync */
r0 = 0x0; dm(DIV4) = R0;
ustat3 = SPEN_A|           /* Enable Channel A */
SLEN32|                    /* 32-bit word length */
FSR|                       /* Frame Sync Required */
SDEN_A;                    /* Enable Channel A DMA */
dm(SPCTL4) = ustat3;
_main.end: jump (pc,0);
```

7 SERIAL PERIPHERAL INTERFACE PORTS

The ADSP-2136x processors are equipped with two synchronous serial peripheral interface ports that are compatible with the industry-standard serial peripheral interface (SPI). Each SPI port also has its own set of registers (the secondary register set contains a B as in `SPIBAUDB`). The SPI ports support communication with a variety of peripheral devices including codecs, data converters, sample rate converters, S/PDIF or AES/EBU digital audio transmitters and receivers, LCDs, shift registers, microcontrollers, and FPGA devices with SPI emulation capabilities.

Features

The processor's SPI ports provide the features and capabilities shown in Table 7-1 and described in the list that follows.

Table 7-1. SPI Port Feature Summary

Feature	SPI	SPIB
Connectivity		
Multiplexed Pinout	No	No
SRU DAI Required	No	Yes
SRU DAI Default Routing	N/A	Yes
Interrupt Default Routing	Yes (P1I)	Yes (P18I)
Protocol		
Master Capable	Yes	Yes
Slave Capable	Yes	Yes

Table 7-1. SPI Port Feature Summary (Cont'd)

Feature	SPI	SPIB
Transmission Simplex	Yes	Yes
Transmission Half Duplex	Yes	Yes
Transmission Full Duplex	Yes (Core and DMA)	Yes (Core and DMA)
Access Type		
Data Buffer	Yes	Yes
Core Data Access	Yes	Yes
DMA Data Access	Yes	Yes
DMA Channels	1	1
DMA Chaining	Yes	Yes
Interrupt Source	Core/DMA	Core/DMA
Boot Capable	Yes	Yes
Local Memory	No	No
Clock Operation	PCLK/4	PCLK/4

- A simple 4-wire interface consisting of two data pins, a device select pin, and a clock pin.
- Special data formats to accommodate little and big endian data, different word lengths, and packing modes.
- Master and multiples slave (multi devices) in which the ADSP-2136x master processor can be connected to up to four other SPI devices.
- Parallel core and DMA access allow full duplex operation.
- Open drain outputs to avoid data contention and to support multimaster scenarios.
- Programmable baud rates, clock polarities, and phases (SPI mode 0–3).

- Master or slave booting from a master SPI device. See "SPI Port Booting" on page 14-38.
- DMA capability to allow transfer of data without core overhead.

Note the SPI interface does not support daisy chain operation, where the `MOSI` and `MISO` pins are internally connected through a FIFO, allowing bypass of data streams.

Pin Descriptions

The SPI protocol uses a 4-wire protocol to enable full-duplex serial communication. Table 7-2 provides detailed pin descriptions and Figure 7-1 on page 7-6 shows the master-slave connections between two devices.

Table 7-2. SPI Pin Descriptions

Internal Node	Type	Description
SPI_CLK_I/O SPIB_CLK_I/O	I/O	**SPI Clock Signal.** This control line is clock driven by the master and regulates the flow of the data bits. The master may transmit data at a varietyof baud rates. The CLK line cycles once for each bit that is transmitted. It is an output signal if the device is configured as a master; it is an input signal if configured as a slave.
SPI_DS_I SPIB_DS_I	I	**SPI Slave Device Select.** This is an active-low input signal that is used to enable slave devices. This signal is like a chip select signal for the slave devices and is provided by the master device. For a master device, it can act as an error input signal in a multi-master environment. In multi-master mode, if the SPIDS input signal of a master is asserted (Low) an error has occurred. This means that another device is also trying to be the master.
SPI_MOSI_I/O SPIB_MOSI_I/O	I/O	**SPI Master Out Slave In.** This data line transmits the output data from the master device and receives the input data to a slave device. This data is shifted out from the MOSI pin of the master and shifted into the MOSI input(s) of the slave(s).

Table 7-2. SPI Pin Descriptions (Cont'd)

Internal Node	Type	Description
SPI_MISO_I/O SPIB_MISO_I/O	I/O	**SPI Master In Slave Out.** This data line transmits the output data from the slave device and receives the input data to the master device. This data is shifted out from the MISO pin of the slave and shifted into the MISO input of the master. There may be no more than one slave that is transmitting data during any particular transfer.
SPIB_FLG3–0_O	O	**SPI Slave Select Out.** The slave select pins are used to address up to 4 slaves in a multi device system. The core flag pins FLAG3–0 can be assigned directly for slave select operation. Note this functionality can also be routed to any of the DAI pins. This frees up the multiplexed core flags for other purposes.
SPIB_CLK_PBEN_O SPIB_MOSI_PBEN_O SPIB_MISO_PBEN_O SPIB_FLG3-0_PBEN_O	O	**SPI Pin Buffer Enable Out Signal.** Only driven in master mode

SRU Programming

The SPIB signals are available through the SRU, and are routed as described in Table 7-3.

Table 7-3. SPI DAI/SRU Signal Connections

Internal Node	DAI Group	SRU Register
Inputs		
SPIB_CLK_I	Group A	SRU_CLK4
SPIB_DS_I SPIB_MOSI_I SPIB_MISO_I	Group E	SRU_EXT_MISCA

Table 7-3. SPI DAI/SRU Signal Connections (Cont'd)

Internal Node	DAI Group	SRU Register
Outputs		
SPIB_CLK_O SPIB_MOSI_O SPIB_MISO_O SPIB_FLG3–0_O	Group D	
SPIB_CLK_PBEN_O SPIB_MOSI_PBEN_O SPIB_MISO_PBEN_O SPIB_FLG3–0_PBEN_O	Group F	

Since the SPI supports a gated clock, it is recommended that programs enable the SPI clock output signal with its related pin buffer enable. This can be done using the macro `SRU (SPI_CLK_PBEN_O, PBEN_03_I)`. If these signals are routed statically high as in `SRU (HIGH, PBEN_03_I)` some SPI timing modes that are based on polarity and phase may not work correctly because the timing is violated.

Functional Description

Each SPI interface contain its own transmit shift (`TXSR, TXSRB`) and receive shift (`RXSR, RXSRB`) registers (not user accessible). The `TXSRx` registers serially transmit data and the `RXSRx` registers receive data synchronously with the SPI clock signal (`SPICLK`). Figure 7-1 shows a block diagram of the SHARC processor SPI interface. The data is shifted into or out of the shift registers on two separate pins: the master in slave out (`MISO`) pin and the master out slave in (`MOSI`) pin.

During data transfers, one SPI device acts as the SPI master by controlling the data flow. It does this by generating the `SPICLK` and asserting the SPI device select signal ($\overline{\texttt{SPIDS}}$). The SPI master receives data using the `MISO` pin and transmits using the `MOSI` pin. The other SPI device acts as the SPI slave by receiving new data from the master into its receive shift register

using the MOSI pin. It transmits requested data out of the transmit shift register using the MISO pin.

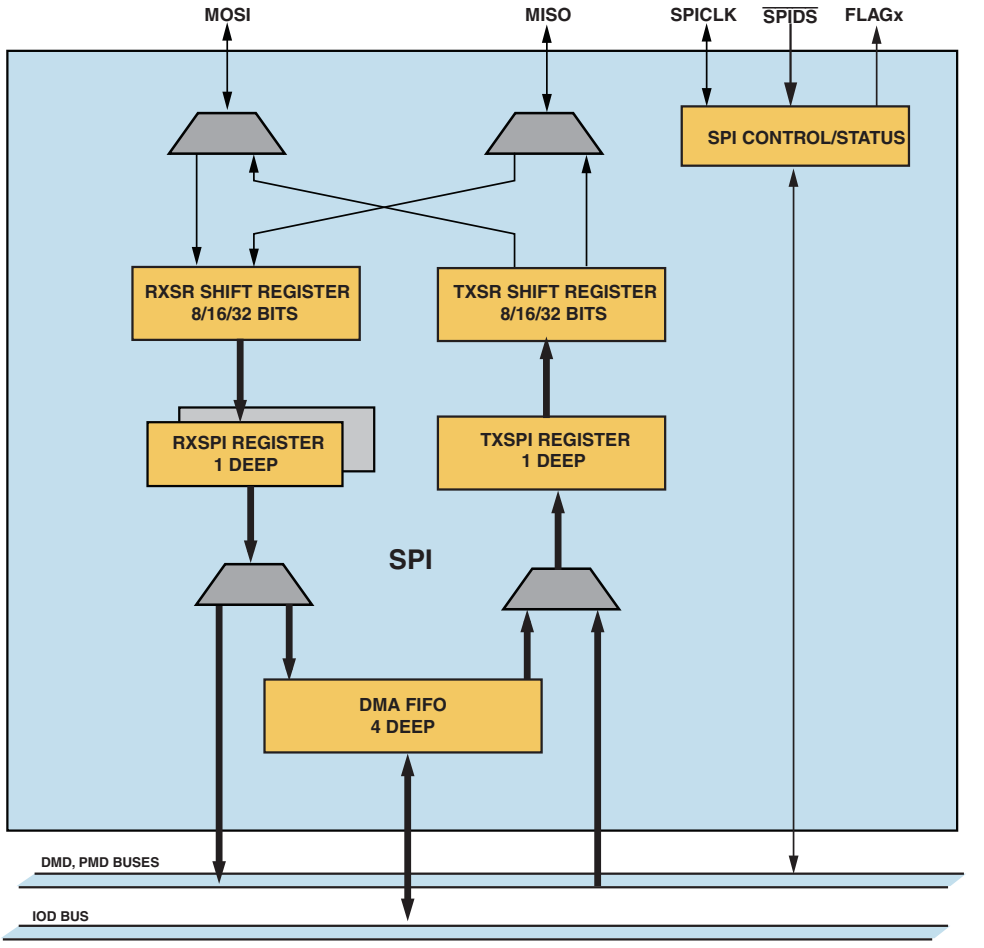


Figure 7-1. SPI Block Diagram

Each SPI port contains a dedicated transmit data buffer (TXSPI, TXSPIB) and a receive data buffer (RXSPI, RXSPIB). Transmitted data is written to TXSPIx and then automatically transferred into the transmit shift register.

Once a full data word has been received in the receive shift register, the data is automatically transferred into RXSPIx, from which the data can be read. When the processor is in SPI master mode, programmable flag pins provide slave selection. These pins are connected to the $\overline{\text{SPIDS}}$ of the slave devices.

The SPI has a single DMA engine which can be configured to support either an SPI transmit channel or a receive channel, but not both simultaneously. Therefore, when configured as a transmit channel, the received data is essentially ignored. When configured as a receive channel, what is transmitted is irrelevant. A 4-word deep FIFO is included to improve throughput on the IOD bus.

The SPI does not support daisy chain operation, whereby the MOSI pins are connected to the MISO pins to form a chain. This operation type requires an internal shift register between the MOSI and MISO pins to bypass the data stream.

Single Master Systems

Figure 7-2 illustrates how the SHARC processor can be used as the slave SPI device. The 16-bit host (A Blackfin ADSP-BF53x processor) is the SPI master. The processor can be booted via its SPI interface to allow application code and data to be downloaded prior to runtime.

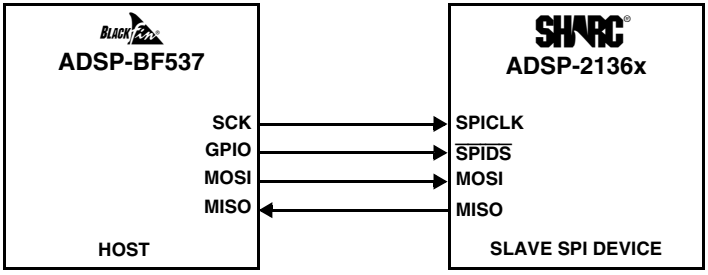


Figure 7-2. SHARC Processor as SPI Slave

Figure 7-3 shows an example SPI interface where the SHARC processor is the SPI master. With the SPI interface, the processor can be directed to alter the conversion resources, mute the sound, modify the volume, and power down the AD1855 stereo DAC.

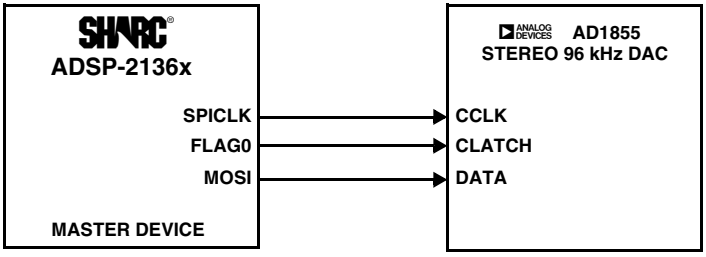


Figure 7-3. SHARC Processor as SPI Master

Broadcast Access

The SPI can also be configured as a master with multiple slaves. In this configuration, no bus mastership scenarios are possible. The master selects the different slaves with the slave select signals to perform data exchange. The SHARC processor is the SPI master.

In broadcast mode, several slaves can be configured to receive from the master, but only one of the slaves can be in transmit mode. This is done by disabling the `MISO` line on the slaves (`DMISO` bit). Only one slave is allowed to communicate back with the master at a time.

Multi Master Systems

The SPI does not have an acknowledgement mechanism to confirm the receipt of data. Without a communication protocol, the SPI master has no knowledge of whether a slave even exists. Furthermore, the SPI has no flow control.

Slaves can be thought of as input/output devices of the master. The SPI does not specify a particular higher-level protocol for bus mastership. In some applications, a higher-level protocol, such as a command-response protocol, may be necessary. Note that the master must initiate the frames for both its' command and the slave's response.

Multi master mode allows an SPI system to transfer mastership from one SPI device to another. In a multi device SPI configuration, several SPI ports are connected and any one (but only one) of them can become a master at any given time.

In this configuration, every MOSI pin in the SPI system is connected. Likewise, every MISO pin in the system is on a single node, and every SPICLK pin should be connected. SPI transmission and reception are always enabled simultaneously, unless the broadcast mode has been selected.

The master's FLAGx pins connect to each of the slave SPI devices in the system via their $\overline{\text{SPIDS}}$ pins. To enable the different slaves, connect the slave $\overline{\text{SPIDS}}$ pins to the programmable flag pins FLAG0-3 of the master SHARC. Since these flags are NOT open drain, slave select pins cannot be shorted together in multi master environment. To control slave selects, an external glue logic is required in a multi-master environment.

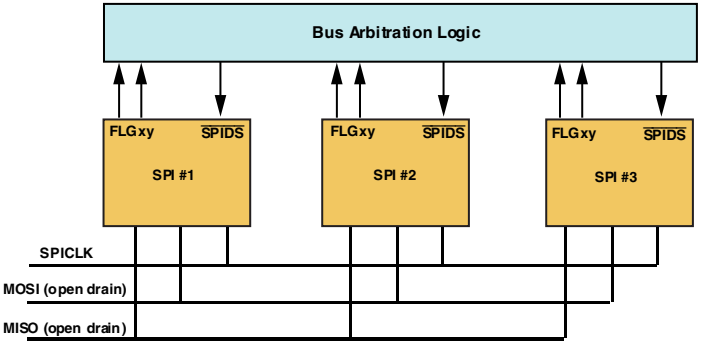


Figure 7-4. Multi Master System

Another feature is implemented to trouble shoot the bus mastership protocol. If a recent SHARC bus master receives an invalidly asserted $\overline{\text{SPIDS}}$ signal, it triggers an error handling scenario using the MME bit (SPIMME bit for DMA) and ISSEN bit to reconfigure the SPI to slave mode, and jump into an ISR. This ensures that any potential driver conflict is solved. For more information, see "Input Slave Select Enable (ISSEN)" on page 7-12.

The following steps show how to implement a system with two SPI devices. Since the slaves cannot initiate transfers over the bus, the master must send frames over the MOSI pin. This ensures that slaves can respond to the bus by sending messages over the MISO pin to the bus master.

1. Slave writes message to its MISO pin.
2. Slave starts polling its $\overline{\text{SPIDS}}$ pin which is currently low.
3. Message is latched by current master and decoded
4. Master de-asserts slave select signal and clears SPIMS bit to become a slave.
5. If bus requester detects the $\overline{\text{SPIDS}}$ pin high, it sets the SPIMS bit to get bus mastership
6. The master selects a slave by driving its' slave select flag pin.

Register Descriptions

This section describes SPICTLx, SPIBAUDx and SPISTATx and SPIDMACx registers as well as some of the primary bit uses in each register.

Control Register (SPICTLx)

The bits in the SPI control registers are used to configure and enable the SPI system. These bits are described in the following sections. For more

information, see “SPI Control Registers (SPICTL, SPICTLB)” on page A-14.

Transfer Initiate Mode (TIMOD)

When the processor is enabled as a master, the initiation of a transfer is defined by the TIMOD bits (1–0). Based on these two bits and the status of the interface, a new transfer is started upon either a read of the RXSPIx registers or a write to the TXSPIx registers. This is summarized in Table 7-4.

Table 7-4. Transfer Initiation

TIMOD	Function	Transfer Initiated Upon	Action, Interrupt
00	Core Receive and Transmit	Initiate new single word transfer upon read of RXSPI and previous transfer completed. The SPICLK is generated after the data is read from the buffer. In this configuration, a dummy read is needed initially to receive all the data transmitted from the transmitter.	The SPI interrupt is latched in every core clock cycle in which the RXSPI buffer has a word in it. Emptying the RXSPI buffer or disabling the SPI port at the same time (SPIEN = 0) stops the interrupt latch.
01	Core Transmit and Receive	Initiate new single word transfer upon write to TXSPI and previous transfer completed.	The SPI interrupt is latched in every core clock cycle in which the TXSPI buffer is empty. Writing to the TXSPI buffer or disabling the SPI port at the same time (SPIEN = 0) stops the interrupt latch.

Table 7-4. Transfer Initiation (Cont'd)

TIMOD	Function	Transfer Initiated Upon	Action, Interrupt
10	Transmit[1] or Receive with DMA	Initiate new multiword transfer upon write to DMA enable bit. Individual word transfers begin with either a DMA write to TXSPI or a DMA read of RXSPI depending on the direction of the transfer as specified by the SPIRCV bit.	If chaining is disabled, the SPI interrupt is latched in the cycle when the DMA count decrements from 1 to 0. If chaining is enabled, interrupt function is based on the PCI bit in the CP register. If PCI = 0, the SPI interrupt is latched at the end of the DMA sequence. If PCI = 1, then the SPI interrupt is latched after each DMA in the sequence. For more information, see "DMA Transfers Between Internal Memory" on page 2-3.
11	Reserved		

1 For transmit DMA, the interrupt occurs when the count reaches zero if INT_ETC (bit 3 of SPIDMAC register) = 0 or when the last bit of the last word is shifted out if INT_ETC = 1.

Input Slave Select Enable (ISSEN)

The behavior of the $\overline{\texttt{SPIDS}}$ input to the SPI module depends on the configuration of the SPI. If the SPI is a slave, $\overline{\texttt{SPIDS}}$ acts as the slave-select input. As master, $\overline{\texttt{SPIDS}}$ can serve as an error-detection input for the SPI in a multi-master environment. The (ISSEN) bit in the SPICTL register enables this feature. When (ISSEN) =1, the $\overline{\texttt{SPIDS}}$ input is the master mode error input; otherwise, $\overline{\texttt{SPIDS}}$ is ignored. The $\overline{\texttt{SPIDS}}$ signal must be connected directly to a chip pin (via the SRU). The state of this input pin must be observable in bit 7 of the SPIFLGx registers.

Disable MISO (DMISO)

Different CPUs or processors can take turns being master, and one master may simultaneously shift data into multiple slaves (broadcast mode). However, only one slave may drive its output to write data back to the master at any given time. This must be enforced in the broadcast mode, where several slaves can be selected to receive data from the master, but

only one slave can be enabled to send data back to the master. The (DMISO) bit disables MISO as an output.

Word Lengths (WL)

The SPI port can transmit and receive the word widths described in the following sections.

8-bit word. Programs can use 8-bit word lengths when transmitting or receiving. When transmitting, the SPI port sends out only the lower eight bits of the word written to the SPI buffer.

For example, if the processor executes the following instructions:

```
r0 = 0x12345678
dm(TXSPI) = r0;
```

the SPI port transmits 0x78.

When receiving, the SPI port packs the 8-bit word to the lower 32 bits of the RXSPI buffer while the upper bits in the registers are zeros.

For example, if an SPI host sends the processor the 32-bit word 0x12345678, the processor receives the following words:

```
0x00000078 //first word
0x00000056 //second word
0x00000034 //third word
0x00000012 //fourth word
```

This code works only if the MSBF bit is zero in both the transmitter and receiver, and the SPICLK frequency is small. If MSBF = 1 in the transmitter and receiver, and SPICLK has a small frequency, the received words follow the order 0x12, 0x34, 0x56, 0x78.

16-bit word. Programs can use 16-bit word lengths when transmitting or receiving. When transmitting, the SPI port sends out only the lower 16 bits of the word written to the SPI buffer.

For example, if the processor executes the following instructions:

```
r0 = 0x12345678
dm(TXSPI) = r0;
```

the SPI port transmits 0x5678.

When receiving, the SPI port packs the16-bit word to the lower 32 bits of the RXSPI buffer while the upper bits in the register are zeros.

For example, if an SPI host sends the processor the 32-bit word 0x12345678, the processor receives the following words:

```
0x00005678  //first word
0x00001234  //second word
```

32-bit word. No packing of the RXSPI or TXSPI registers is necessary as the entire 32-bit register is used for the data word.

Modes (CLKPL, CPHASE)

The SPI supports four different combinations of serial clock phases and polarity called SPI modes. The application code can select any of these combinations using the CLKPL and CPHASE bits (10 and 11).

Figure 7-5 on page 7-16 shows the transfer format when CPHASE = 0 and CPHASE = 1. Each diagram shows two waveforms for SPICLK—one for CLKPL = 0 and the other for CLKPL = 1. The diagrams may be interpreted as master or slave timing diagrams since the SPICLK, MISO, and MOSI pins are directly connected between the master and the slave. The MISO signal is the output from the slave (slave transmission), and the MOSI signal is the output from the master (master transmission).

The SPICLK signal is generated by the master, and the $\overline{\text{SPIDS}}$ signal represents the slave device select input to the processor from the SPI master. The diagrams represent 8-bit transfers (WL = 0) with MSB first (MSBF = 1). Any combination of the WL and MSBF bits of the SPICTL register is allowed.

For example, a 16-bit transfer with the LSB first is one possible configuration.

The clock polarity and the clock phase should be identical for the master device and slave devices involved in the communication link. The transfer format from the master may be changed between transfers to adjust to various requirements of a slave device.

When CPHASE = 0, the slave-select line, $\overline{\text{SPIDS}}$, must be inactive (HIGH) between each word in the transfer. Even in SPI slave mode when CPHASE = 0, the master should de assert the $\overline{\text{SPIDS}}$ line between each transfer. When CPHASE = 1, $\overline{\text{SPIDS}}$ may either remain active (LOW) between successive transfers or be inactive (HIGH).

Figure 7-5 shows the SPI transfer protocol for CPHASE = 0. Note that SPICLK starts toggling in the middle of the data transfer where the bit settings are WL = 0, and MSBF = 1.

Figure 7-5 also shows the SPI transfer protocol for CPHASE = 1. Note that SPICLK starts toggling at the beginning of the data transfer where the bit settings are WL = 0, and MSBF = 1.

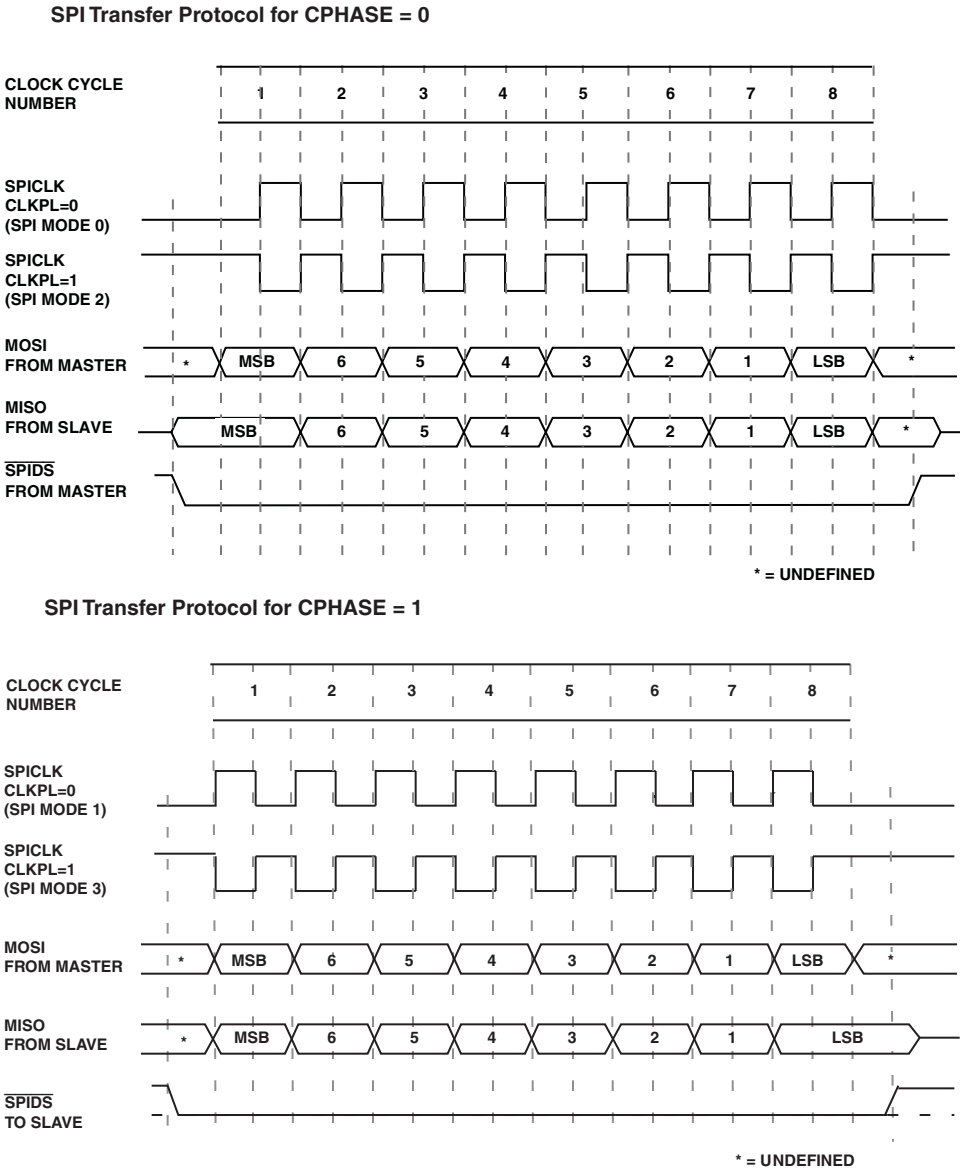

SPI Transfer Protocol for CPHASE = 0
CLOCK CYCLE NUMBER
1 2 3 4 5 6 7 8
SPICLK CLKPL=0 (SPI MODE 0)
SPICLK CLKPL=1 (SPI MODE 2)
MOSI FROM MASTER
* MSB 6 5 4 3 2 1 LSB *
MISO FROM SLAVE
MSB 6 5 4 3 2 1 LSB *
SPIDS FROM MASTER
* = UNDEFINED
SPI Transfer Protocol for CPHASE = 1
CLOCK CYCLE NUMBER
1 2 3 4 5 6 7 8
SPICLK CLKPL=0 (SPI MODE 1)
SPICLK CLKPL=1 (SPI MODE 3)
MOSI FROM MASTER
* MSB 6 5 4 3 2 1 LSB *
MISO FROM SLAVE
* MSB 6 5 4 3 2 1 LSB
SPIDS TO SLAVE
* = UNDEFINED

Figure 7-5. SPI Transfer Protocol for CPHASE = 0

Open Drain Mode (OPD)

In a multimaster or multislave SPI system, the data output pins (MOSI and MISO) can be configured to behave as open drain drivers to prevent contention and possible damage to pin drivers. An external pull-up resistor is required on both the MOSI and MISO pins when this option is selected.

When the OPD bit (13) is set and the SPI ports are configured as masters, the MOSI pin is three-stated when the data driven out onMOSI is logic-high. The MOSI pin is not three-stated when the driven data is logic-low. A zero is driven on the MOSI pin in this case. Similarly, when OPD is set and the SPI ports are configured as slaves, the MISO pin is three-stated if the data driven out on MISO is logic-high.

Normally, it is acceptable to enable (SRU) output buffers to permanently assert the PBEN_I signals. However, this does not work for the open drain mode, because the DAI buffer always actively drives the output pin. Where open drain mode is used for the MOSI/MISO pins, programs must first connect the PBEN_I signals to the pin enable associated with the SPIB SRU buffers. Programs must then tie the pin input signal to ground. For more information, see Chapter 5, Digital Application Interface.

Enable (SPIEN)

For SPI slaves, the slave-select input acts like a reset for the internal SPI logic. For this reason, the $\overline{\text{SPIDS}}$ line must be error free. The SPIEN signal can also be used as a software reset of the internal SPI logic. An exception to this is the W1C-type (write 1-to-clear) bits in the SPISTATx registers. These bits remain set if they are already set.

Always clear the W1C-type bits before re-enabling the SPI, as these bits do not get cleared even if SPI is disabled. This can be done by writing 0xFF to the SPISTATx registers. In the case of an MME error, enable the SPI ports after $\overline{\text{SPIDS}}$ is deasserted.

Packing (PACKEN)

In order to communicate with 8-bit SPI devices and store 8-bit words in internal memory, a packed transfer feature is built into the SPI port. Packing is enabled through the PACKEN bit (15). The SPI unpacks data when it transmits and packs data when it receives. When packing is enabled, two 8-bit words are packed into one 32-bit word. When the SPI port is transmitting, two 8-bit words are unpacked from one 32-bit word. When receiving, words are packed into one 32-bit word from two 8-bit words.

An example of unpacking data before transmitting follows.

The value 0xXXLMXXJK (where XX is any random value and JK and LM are data words to be transmitted out of the SPI port) is written to the TXSPI register. The processor transmits 0xJK first and then transmits 0xLM.

An example of packing on the received data:

The receiver packs the two words received, 0xJK and then 0xLM, into a 32-bit word. They appear in the RXSPI register as:

- 0x00LM00JK => if SGN is configured to 0 or L, J < 7
  0xFFLMFFJK => if SGN is configured to 1 and L, J > 7

Status Register (SPISTATx)

This bits in this register (MME, TUNF and ROVF) provide information on transmission errors for the port and are described in the following sections. Corresponding bits (SPIMME, SPIUNF and SPIOVF) in the SPIDMACx registers are set when an error occurs during a DMA transfer. These sticky bits generate an SPI interrupt when any one of them are set.

Always clear the W1C-type bits before re-enabling the SPI, as these bits are not cleared even if SPI is disabled. This can be done by writing 0xFF to the SPISTATx registers. In the case of an MME error, enable the SPI ports after $\overline{\text{SPIDS}}$ is deasserted.

Multi Master Error (MME)

The MME bit (1) is set when the $\overline{\text{SPIDS}}$ input pin of a device that is enabled as a master is driven low by some other device in the system. This occurs in multimaster systems when another device is also trying to be the master.

To enable this feature, set the ISSEN bit in the SPICTL register. As soon as this error is detected, the following actions are taken:

1. The SPIMS control bit in SPICTL is cleared, configuring the SPI interface as a slave.
2. The SPIEN control bit in SPICTL is cleared, disabling the SPI system.
3. The MME status bit in SPISTAT is set.
4. An SPI interrupt is generated.

These four conditions persist until the MME bit is cleared by a write 1-to-clear (W1C-type) software operation. Until the MME bit is cleared, the SPI cannot be re-enabled, even as a slave. Hardware prevents the program from setting either SPIEN or SPIMS while MME is set.

When MME is cleared, the interrupt is deactivated. Before attempting to re-enable the SPI as a master, the state of the $\overline{\text{SPIDS}}$ input pin should be checked to ensure that it is high; otherwise, once SPIEN and SPIMS are set, another mode-fault error condition will immediately occur. The state of the input pin is reflected in the input slave select status bit (bit 7) in the SPIFLG register.

As a result of SPIEN and SPIMS being cleared, the SPI data and clock pin drivers (MOSI, MISO, and SPICLK) are disabled. However, the slave-select output pins revert to control by the processor flag I/O module registers. This may cause contention on the slave-select lines if these lines are still being driven by the processor. In order to ensure that the slave-select output drivers are disabled once a MME error occurs, the program must configure these pins as inputs by setting (= 1) the flag output select bits,

FLAG3-0, in the FLAGS register prior to configuring the SPI port. See the FLAGs value register description in the *SHARC Processor Programming Reference* “Registers” appendix.

Transmission Error Bit (TUNF)

The TUNF bit (2) is set when all of the conditions of transmission are met and there is no new data in TXSPI (TXSPI is empty). In this case, the transmission contents depend on the state of the SENDZ bit in the SPICTL register. The TUNF bit is cleared by a W1C-type software operation.

Reception Error Bit (ROVF)

The ROVF flag (bit 4) is when a new transfer has completed before the previous data is read from the RXSPI register. This bit indicates that a new word was received while the receive buffer was full. The ROVF flag is cleared by a W1C-type software operation. The state of the GM bit in the SPICTL register determines whether the RXSPI register is updated with the newly received data or whether that new data is discarded.

Transmit Collision Error Bit (TXCOL)

The TXCOL flag (bit 6) is set when a write to the TXSPI register coincides with the load of the shift register. The write to TXSPI can be via the software or the DMA. This bit indicates that corrupt data may have been loaded into the shift register and transmitted. In this case, the data in TXSPI may not match what was transmitted. This error can easily be avoided by proper software control. The TXCOL bit is cleared by a W1C-type software operation.

This bit is never set when the SPI is configured as a slave with CPHASE = 0. The collision may occur, but it cannot be detected.

BAUD Rate Register (SPIBAUDx)

For master devices, the clock rate is determined by the 15-bit value of the baud rate registers (SPIBAUDx) as shown in Table 7-5. For slave devices, the value in the SPIBAUDx register is ignored. For more information, see "SPI Baud Rate Registers (SPIBAUD, SPIBAUDB)" on page A-20.

Table 7-5. SPI BAUD Rate Settings

BAUDR Bit Setting	Divider	SPICLK (PCLK = 167 MHz)
0	N/A	N/A
1	8	41.7 MHz
2	16	20.8 MHz
3	24	13.9 MHz
4	32	10.4 MHz
32,767 (0x7FFF)	262136	1.3 KHz

DMA Control Register (SPIDMACx)

This register contains the error status bits (SPIMME, SPIUNF, SPIOVF and SPIERRS). These bits are set when an error occurs during a DMA transfer. The (SPIERRS) bit is set if any of the SPIMME, SPIUNF, SPIOVF bits is set. An interrupt can be generated with the (INTERR) bit to respond to these errors.

Data Transfer Types

The SPI is capable of transferring data via the core and DMA. The following sections describe these transfer types.

Core Transfers

For core-driven SPI transfers, the software has to read from or write to the RXSPIx and TXSPIx registers respectively to control the transfer. It is important to check the buffer status before reading from or writing to these registers because the core *does not* hang when it attempts to read from an empty buffer or write to a full buffer. When the core writes to a full buffer, the data that is in that buffer is overwritten and the SPI begins transmitting the new data. Invalid data is obtained when the core reads from an empty buffer.

For a master, when the transmit buffer becomes empty, or the receive buffer becomes full, the SPI device stalls the SPI clock until it reads all the data from the receive buffer or it detects that the transmit buffer contains a piece of data.

- When a master is configured with TIMOD = 01 and the transmit buffer becomes empty, the SPI device stalls the SPI clock until a piece of data is written to the transmit buffer.
- When a master is configured with TIMOD = 00 and the receive buffer becomes full, the SPI device stalls the SPI clock until all of the data is read from the receive buffer.

DMA Transfers

The SPI ports support both master and slave mode DMA. The following sections describe slave and master mode DMA operations, DMA chaining, switching between transmit and receive DMA operations, and processing

DMA interrupt errors. Guidelines that programs should follow when performing DMA transfers over the SPI include:

- Do not write to the TXSPIx registers during an active SPI transmit DMA operation because DMA data will be overwritten.
- Similarly, do not read from the RXSPIx registers during active SPI DMA receive operations.
- Writes to the TXSPIx registers during an active SPI receive DMA operation are permitted. The RXS register is cleared when the RXSPIx registers are read.
- Reads from the RXSPIx registers are allowed at any time during transmit DMA.
- Interrupts are generated based on DMA events and are configured in the SPIDMACx registers.

To avoid data corruption, enable the SPI port before enabling DMA.

In order for a transmit DMA operation to begin, the transmit buffer must initially be empty (TXS = 0). While this is normally the case, this means that the TXSPIx registers should not be used for any purpose other than SPI transfers. For example, the TXSPIx registers should not be used as a scratch register for temporary data storage. Writing to the TXSPIx registers via the software sets the TXS bit.

For receive DMA in master mode the SPICLK stops only when the FIFO and RXSPI buffer is full (even if the DMA count is zero). Therefore, SPICLK runs for an additional five word transfers filling junk data in the FIFO and the RXSPIx buffers. This data must be cleared before a new DMA is initiated.

Slave DMA Transfer Preparation

When enabled as a slave, the device prepares for a new transfer according to the function and actions described in Table 7-4 on page 7-11.

The following steps illustrate the SPI receive or transmit DMA sequence in an SPI slave in response to a master command:

1. Once the slave-select input is active, the processor starts receiving and transmitting data on active SPICLK edges. The data for one channel (TX or RX) is automatically transferred from/to memory by the DMA controller. The function of the other channel is dependant on the GM and SENDZ bits in the SPICTL register.
2. Reception or transmission continues until the SPI DMA word count register transitions from 1 to 0.
3. A number of conditions can occur while the processor is configured for the slave mode:
    - If the DMA engine cannot keep up with the receive data stream during receive operations, the receive buffer operates according to the state of the GM bit in the SPICTLx registers.
    - If GM = 0 and the DMA buffer is full, the incoming data is discarded, and the RXSPIx register is not updated. While performing a receive DMA, the transmit buffer is assumed to be empty. If SENDZ = 1, the device repeatedly transmits zeros on the MISO pin. If SENDZ = 0, it repeatedly transmits the contents of the TXSPIx registers.
    - If GM = 1 and the DMA buffer is full, the device continues to receive new data from the MOSI pin, overwriting the older data in the DMA buffer.
    - If the DMA engine cannot keep up with the transmit data stream during a transmit operation because another DMA engine has been granted the bus (or for another reason), the

transmit port operates according to the state of the SENDZ bit in the SPICTLx registers.

If SENDZ = 1 and the DMA buffer is empty, the device repeatedly transmits zeros on the MISO pin. If SENDZ = 0 and the DMA buffer is empty, it repeatedly transmits the last word transmitted before the DMA buffer became empty. All aspects of SPI receive operation should be ignored. The data in the RXSPIx registers is not intended to be used, and the RXS and ROVF bits should be ignored. The ROVF overrun condition cannot generate an error interrupt in this mode.

While a DMA transfer may be used on one channel (TX or RX), the core (based on the RXS and TXS status bits) can transfer data in the other direction.

DMA Chaining

The serial peripheral interfaces support both single and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain. For more information, see “SPI TCB” on page 2-31.

Setting Up and Starting Chained DMA

Configuring and starting chained DMA transfers over the SPI port is the same as for the serial port, with one exception. Contrary to SPORT DMA chaining, (where the first DMA in the chain is configured by the first TCB), for SPI DMA chaining, the first DMA is not initialized by a TCB. Instead, the first DMA in the chain must be loaded into the SPI parameter registers (IISPI, IMSPI, CSPI, IISPIB, IMSPIB, CSPIB), and the chain pointer registers (CPSPI, CPSPIB) point to a TCB that describes the second DMA in the sequence. Table 2-15 on page 2-31 shows the order of register loading.

Writing an address to the CPSPIx, registers does not begin a chained DMA sequence unless the IISPI, IMSPI, CSPI, IISPIB, IMSPIB, and CSPIB registers are initialized, SPI DMA is enabled, the SPI port is enabled, and SPI DMA chaining is enabled.

Core and DMA Transfers

When the SPI DMA engine is configured for transmitting:

1. The receive interface cannot generate an interrupt, but the status can be polled.
2. The four-deep FIFO is not available in the receive path.

Similarly, when the SPI DMA engine is configured for receiving:

1. The transmit interface cannot generate an interrupt, but the status can be polled.
2. The four-deep FIFO is not available in the transmit path.

Changing Configuration

Programs should take the following precautions when changing SPI configurations.

- The SPI configuration must not be changed during a data transfer.
- Change the clock polarity only when no slaves are selected.
- Change the SPI configuration when SPIEN = 0. For example, if operating as a master in a multislave system, and there are slaves that require different data or clock formats, then the master SPI should be disabled, reconfigured, and then re-enabled.

However, when an SPI communication link consists of:

1. A single master and a single slave,
2. CPHASE = 1
3. The slave's slave select input is tied low

Then the program can change the SPI configuration. In this case, the slave is always selected. Data corruption can be avoided by enabling the slave only after configuring both the master and slave devices.

Starting and Stopping Data Transfers

An SPI transfer's defined start and end depend on whether the device is configured as a master or a slave, whether CPHASE mode is selected, and whether the transfer initiation mode is (TIMOD) selected. For a master SPI with CPHASE = 0, a transfer starts when either the TXSPI register is written or the RXSPI register is read, depending on the TIMOD selection. At the start of the transfer, the enabled slave-select outputs are driven active (low). However, the SPICLK starts toggling after a delay equal to one-half (0.5) the SPICLK period. For a slave with CPHASE = 0, the transfer starts as soon as the $\overline{\text{SPIDS}}$ input transitions to low.

For CPHASE = 1, a transfer starts with the first active edge of SPICLK for both slave and master devices. For a master device, a transfer is considered complete after it sends and simultaneously receives the last data bit. A transfer for a slave device is complete after the last sampling edge of SPICLK.

The RXS bit defines when the receive buffer can be read. The TXS bit defines when the transmit buffer can be filled. The end of a single word transfer occurs when the RXS bit is set. This indicates that a new word has been received and latched into the receive buffer, RXSPI. The RXS bit is set shortly after the last sampling edge of SPICLK. There is a 4 PCLK cycle latency for a master/slave device, depending on synchronization. This is independent of CPHASE, TIMOD and the baud rate.

To maintain software compatibility with other SPI devices (HC-11), the SPI transfer finished bit (SPIF) is also available for polling. This bit may have slightly different behavior from that of other commercially available devices. For a slave device, SPIF is set at the same time as RXS. For a master device, SPIF is set one-half (0.5) of the SPICLK period after the last SPICLK edge, regardless of CPHASE or CLKPL. The baud rate determines when the SPIF bit is set. In general, SPIF is set after RXS, but at the lowest baud rate settings (SPIBAUD < 4). The SPIF bit is set before the RXS bit, and consequently before new data has been latched into the RXSPI buffer. Therefore, for SPIBAUD = 2 or SPIBAUD = 3, the processor must wait for the RXS bit to be set (after SPIF is set) before reading the RXSPI buffer. For larger SPIBAUD settings (SPIBAUD > 4), RXS is set before SPIF.

Interrupts

The following section describes SPI operations using both the core and direct memory access (DMA).

Interrupt Sources

The SPI ports can generate interrupts in five different situations. During core-driven transfers, an SPI interrupt is triggered:

1. When the TXSPI buffer has the capacity to accept another word from the core
2. When the RXSPI buffer contains a valid word to be retrieved by the core

The TIMOD (transfer initiation and interrupt) register determines whether the interrupt is based on the TXSPI or RXSPI buffer status.

If configured to generate an interrupt when SPIRX is full (TIMOD = 00), the interrupt will be active 1 PCLK cycle after the RXS bit is set.

During DMA driven transfers, an SPI interrupt is triggered:

1. At the completion of a single DMA transfer
2. At the completion of a number of DMA sequences (if DMA chaining is enabled)
3. When a DMA error has occurred

Note that the SPIDMAC register must be initialized properly to enable DMA interrupts.

Each of these five interrupts are serviced using the interrupt associated with the module being used. The primary SPI uses the SPIHI interrupt and the secondary SPI uses the SPILI interrupt. Whenever an SPI interrupt occurs (regardless of the cause), the SPILI or SPIHI interrupts are latched. To service the primary SPI port, unmask (set = 1) the SPIHI bit (bit 12) in the IMASK register. To service the secondary SPI port, unmask (set = 1) the SPILIMSK bit (bit 19) in the LIRPTL register. For a list of these bits, see "Interrupts" in Appendix B, Interrupts

When using DMA transfers, programs must also specify whether to generate interrupts based on transfer or error status. For DMA transfer status based interrupts, set the INTEN bit in the SPIDMAC register; otherwise, set the INTERR bit to trigger the interrupt if one of the error conditions occurs during the transmission like multimaster error (MME), transmit buffer underflow (TUNF – only if SPIRCV = 0), or receive buffer overflow (ROVF – only if SPIRCV = 1).

During core-driven transfers, the TUNF and ROVF error conditions do not generate interrupts.

When DMA is disabled, the processor core may read from the RXSPI register or write to the TXSPI data buffer. The RXSPI and TXSPI buffers are memory-mapped IOP registers. A maskable interrupt is generated when the receive buffer is not empty or the transmit buffer is not full. The TUNF and ROVF error conditions do not generate interrupts in these modes.

Internal Transfer Completion

The DMA interrupts indicate DMA completion status and DMA error status. These interrupts are latched in the core when the DMA count reaches zero.

For a chained DMA of blocks (PCI = 1), the interrupt is generated whenever the DMA count reaches zero

DMA Error Interrupts

The SPIUNF and SPIOVF bits of the SPIDMACx registers indicate transmission errors during a DMA operation in slave mode. When one of the bits is set, an SPI interrupt occurs. The following sequence details the steps to respond to this interrupt.

With disabling the SPI:

1. Disable the SPI port by writing 0x00 to the SPICTLx registers.
2. Disable DMA and clear the FIFO by writing 0x80 to the SPIDMACx registers. This ensures that any data from a previous DMA operation is cleared before configuring a new DMA operation.
3. Clear all errors by writing to the W1C-type bits in the SPISTATx registers. This ensures that the error bits SPIOVF and SPIUNF (in the SPIDMACx registers) are cleared when a new DMA is configured.
4. Reconfigure the SPICTLx registers and enable the SPI using the SPIEN bit.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMACx registers.

Without disabling the SPI:

1. Disable DMA and clear the FIFO by writing 0x80 to the SPIDMAC register. This ensures that any data from a previous DMA operation is cleared before configuring a new DMA operation.
2. Clear the RXSPIx/TXSPIx registers and the buffer status without disabling SPI. This can be done by ORing 0xc0000 with the present value in the SPICTLx registers. Use the RXFLSH and TXFLSH bits to clear the RXSPIx/TXSPIx registers and the buffer status.
3. Clear all errors by writing to the W1C-type bits in the SPISTAT register. This ensures that error bits SPIOVF and SPIUNF in the SPIDMACx registers are cleared when a new DMA is configured.
4. Reconfigure the SPICTL register to remove the clear condition on the RXSPI/TXSPI register bits.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMACx register.

Slave Select Timing

When the processor is configured as an SPI slave, the SPI master must drive an SPICLK signal that conforms with Figure 7-6. For exact timing parameters, please refer to the *ADSP-2136x SHARC Processor Data Sheet*.

As shown in Figure 7-6, the $\overline{\text{SPIDS}}$ lead time (T1), the $\overline{\text{SPIDS}}$ lag time (T2), and the sequential transfer delay time (T3) must always be greater than or equal to one-half the SPICLK period. The minimum time between successive word transfers (T4) is two SPICLK periods. This time period is measured from the last active edge of SPICLK of one word to the first active edge of SPICLK of the next word. This calculation is independent from the configuration of the SPI (CPHASE, SPIMS, and so on).

This is shown as:
T3 = 0.5 SPI clock period
T4 = 1.5 SPI clock period + T3

For a master device with CPHASE = 0 or CPHASE = 1, this means that the slave-select output is inactive (high) for at least one-half the SPICLK period. In this case, T1 and T2 are each always be equal to one-half the SPICLK period.

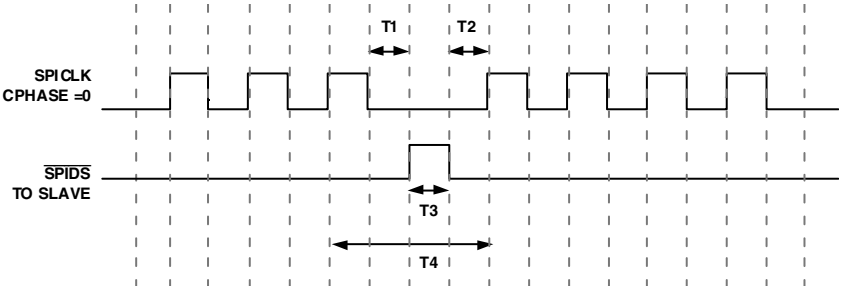


Figure 7-6. SPICLK Timing

Debug Features

The following sections provide information on features that help in debugging SPI software.

Shadow Register

A shadow register for the receive data buffer, SPIRX, is available for use in debugging software. This register, SPIRX_SHADOW, is at a different address from SPIRX, but its contents are identical to that of SPIRX. When SPIRX is read from core, the RXS bit is cleared and an SPI transfer may be initiated (if TIMOD = 00). No such hardware action occurs when the shadow register is read. SPIRX_SHADOW is a read-only register and only accessible by the core.

Internal Loopback Mode

In this mode different types of loopback are possible since there is only one DMA channel available:

- Core receive and transmit transfers
- Transmit DMA and core receive transfers
- Core Transmit and DMA receive transfers

To loop data back from MOSI to MISO, the MISO pin is internally disconnected. The MOSI pin will contain the value being looped back. Programs should set the SPIEN, SPIMS, and ILPBK bits in the SPICTLx register.

Loopback operation is only used in master mode.

Loopback Routing

The SPI supports an internal loopback mode using the SRU. For more information, see “Loop Back Routing” on page 5-30.

Programming Model

The section describes which sequences ofsoftware steps are required to get the peripheral working successfully.

Master Mode Core Transfers

When the SPI is configured as a master, the SPI ports should be configured and transfers started using the following steps:

1. When CPHASE is set to 0 with CPHASE = 1, the slave-selects are automatically controlled by the SPI port. When CPHASE = 1, the slave-selects are controlled by the core, and the user software has to control the pins through the SPIFLGx bits. Before enabling the SPI

port, programs should specify which of the slave-select signals to use, setting one or more of the required SPI flag select bits (DSxEN) in the SPIFLGx registers.

2. Write to the SPICTLx and SPIBAUDx registers, enabling the device as a master and configuring the SPI system by specifying the appropriate word length, transfer format, baud rate, and other necessary information.

3. If CPHASE = 1 (user-controlled, slave-select signals), activate the desired slaves by clearing one or more of the SPI flag bits (SPIFLGx) in the SPIFLGx registers.

4. Initiate the SPI transfer. The trigger mechanism for starting the transfer is dependant upon the TIMOD bits in the SPICTLx registers. See Table 7-4 on page 7-11 for details.

5. The SPI generates the programmed clock pulses on SPICLK. The data is shifted out of MOSI and shifted in from MISO simultaneously. Before starting to shift, the transmit shift register is loaded with the contents of the TXSPIx registers. At the end of the transfer, the contents of the receive shift register are loaded into the RXSPIx registers.

6. With each new transfer initiate command, the SPI continues to send and receive words, according to the SPI transfer mode (TIMOD bit in SPICTLx registers). See Table 7-4 on page 7-11 for more details.

If the transmit buffer remains empty, or the receive buffer remains full, the device operates according to the states of the SENDZ and GM bits in the SPICTLx registers.

- If SENDZ = 1 and the transmit buffer is empty, the device repeatedly transmits zeros on the MOSI pin. One word is transmitted for each new transfer initiate command.

- If SENDZ = 0 and the transmit buffer is empty, the device repeatedly transmits the last word transmitted before the transmit buffer became empty.
- If GM = 1 and the receive buffer is full, the device continues to receive new data from the MISO pin, overwriting the older data in the RXSPI buffer.
- If GM = 0 and the receive buffer is full, the incoming data is discarded, and the RXSPI register is not updated.

Slave Mode Core Transfers

When a device is enabled as a slave (and DMA mode is not selected), the start of a transfer is triggered by a transition of the $\overline{\text{SPIDS}}$ select signal to the active state (LOW) or by the first active edge of the clock (SPICLK), depending on the state of CPHASE.

The following steps illustrate SPI operation in slave mode.

1. Write to the SPICTLx registers to make the mode of the serial link the same as the mode that is set up in the SPI master.
2. Write the data to be transmitted into the TXSPIx registers to prepare for the data transfer.
3. Once the $\overline{\text{SPIDS}}$ signal's falling edge is detected, the slave starts sending and receiving data on active SPICLK edges.
4. The reception or transmission continues until $\overline{\text{SPIDS}}$ is released or until the slave has received the proper number of clock cycles.
5. The slave device continues to receive or transmit with each new falling-edge transition on $\overline{\text{SPIDS}}$ or active SPICLK clock edge.

If the transmit buffer remains empty, or the receive buffer remains full, the devices operate according to the states of the SENDZ and GM bits in the SPICTLx registers.

- If SENDZ = 1 and the transmit buffer is empty, the device repeatedly transmits zero's on the MISO pin.
- If SENDZ = 0 and the transmit buffer is empty, it repeatedly transmits the last word transmitted before the transmit buffer became empty.
- If GM = 1 and the receive buffer is full, the device continues to receive new data from the MOSI pin, overwriting the older data in the RXSPI buffer.
- If GM = 0 and the receive buffer is full, the incoming data is discarded, and the RXSPIx registers are not updated.

Master Mode DMA Transfers

To configure the SPI port for master mode DMA transfers:

1. Specify which FLAG pins to use as the slave-select signals by setting one or more of the DSxEN bits (bits 3–0) in the SPI flag (SPIFLGx) registers.
2. Enable the device as a master and configure the SPI system by selecting the appropriate word length, transfer format, baud rate, and so on in the SPIBAUDx and SPICTLx registers. The TIMOD field (bits 1–0) in the SPICTLx registers is configured to select transmit or receive with DMA mode (TIMOD = 10).
3. Activate the desired slaves by clearing one or more of the SPI flag bits (SPIFLGx) of the SPIFLGx registers, if CPHASE = 1.

4. For a single DMA, define the parameters of the DMA transfer by writing to the IISPIx, IMSPIx, and CSPIx registers. For DMA chaining, write the chain pointer address to the CPSPIx registers.

Write to the SPI DMA configuration registers, (SPIDMACx), to specify the DMA direction (SPIRCV, bit 1) and to enable the SPI DMA engine (SPIDEN, bit 0). If DMA chaining is desired, set (= 1) the SPICHEN bit (bit 4) in the SPIDMACx registers.

If flags are used as slave selects, programs should activate the flags by clearing the flag after SPICTLx and SPIBAUDx are configured, but before enabling the DMA. When CPHASE = 0, or CPHASE = 1, the flags are automatically activated by the SPI ports.

When enabled as a master, the DMA engine transmits or receives data as follows.

1. If the SPI system is configured for transmitting, the DMA engine reads data from memory into the SPI DMA FIFO. Data from the DMA FIFO is loaded into the TXSPIx registers and then into the transmit shift register. This initiates the transfer on the SPI port.

2. If configured to receive, data from the RXSPIx registers is automatically loaded into the SPI DMA FIFO. Then the DMA engine reads data from the SPI DMA FIFO and writes to memory. Finally, the SPI initiates the receive transfer.

3. The SPI generates the programmed signal pulses on SPICLK and the data is shifted out of MOSI and in from MISO simultaneously.

4. The SPI continues sending or receiving words until the SPI DMA word count register transitions from 1 to 0.

If the DMA engine is unable to keep up with the transmit stream during a transmit operation because the IOP requires the IOD (I/O data) bus to service another DMA channel (or for another reason), the SPICLK stalls until data is written into the TXSPI register. All aspects of SPI receive

operation should be ignored. The data in the RXSPI register is not intended to be used, and the RXS (bits 28–27 and 31–30 in the SPICTLx registers) and SPISTAT bits (bits 26 and 29) should be ignored. The ROVF overrun condition cannot generate an error interrupt in this mode.

If the DMA engine cannot keep up with the receive data stream during receive operations, then SPICLK stalls until data is read from RXSPI. While performing a receive DMA, the processor core assumes the transmit buffer is empty. If SENDZ = 1, the device repeatedly transmits 0s. If SENDZ = 0, it repeatedly transmits the contents of the TXSPI register. The TUNF underrun condition cannot generate an error interrupt in this mode.

A master SPI DMA sequence may involve back-to-back transmission and/or reception of multiple chained DMA transfers. The SPI controller supports such a sequence with minimal processor core interaction.

Slave Mode DMA Transfers

A slave mode DMA transfer occurs when the SPI port is enabled and configured in slave mode, and DMA is enabled. When the $\overline{\text{SPIDS}}$ signal transitions to the active-low state or when the first active edge of SPICLK is detected, it triggers the start of a transfer.

When the SPI is configured for receive/transmit DMA, the number of words configured in the DMA count register should match the actual data transmitted. When the SPI DMA is used, the internal DMA request is generated for a DMA count of four. In case the count is less than four, one DMA request is generated for all the bytes. For example, when a DMA count of 16 is programmed, four DMA requests are generated (that is, four groups of four). For a DMA count of 18, five DMA requests are generated (four groups of four and one group of two). In case the SPI DMA is programmed with a value more than the actual data transmitted, some bytes may not be received by the SPI DMA due to the condition for generating the DMA request.

To configure for slave mode DMA:

1. Write to the SPICTLx register to make the mode of the serial link the same as the mode that is set up in the SPI master. Configure the TIMOD field to select transmit or receive DMA mode (TIMOD = 10).
2. Define DMA receive (or transmit) transfer parameters by writing to the IISPIx, IMSPIx, and CSPIx registers. For DMA chaining, write to the chain pointer address of the CPSPIx registers.
3. Write to the SPIDMACx registers to enable the SPI DMA engine and configure the following:
    - A receive access (SPIRCV = 1) or
    - A transmit access (SPIRCV = 0)
    - If DMA chaining is desired, set the SPICHEN bit in the SPIDMACx registers.

Enable the SPI port before enabling DMA to avoid data corruption.

Chained DMA Transfers

The sequence for setting up and starting a chained DMA is outlined in the following steps.

1. Clear the chain pointer register.
2. Configure the TCB associated with each DMA in the chain except for the first DMA in the chain.
3. Write the first three parameters for the initial DMA to the IISPI, IMSPI, CSPI, IISPIB, IMSPIB, and CSPIB registers directly.
4. Select a baud rate using the SPIBAUD register.

5. Select which flag to use as the SPI slave select signal in the SPIFLG register.
6. Configure and enable the SPI port with the SPICTL, SPICTLB registers.
7. Configure the DMA settings for the entire sequence, enabling DMA and DMA chaining in the SPIDMAC register.

Begin the DMA by writing the address of a TCB (describing the second DMA in the chain) to the CPSPI, CPSPI registers.

Stopping Core Transfers

When performing transmit operations with the SPI port, disabling the SPI port prematurely can cause data corruption and/or not fully transmitted data. Before the program disables the SPI port in order to reconfigure it, the status bits should be polled to ensure that all valid data has been completely transferred. For core-driven transfers, data moves from the TXSPI buffer into a shift register. The following bits should be checked before disabling the SPI port:

1. Wait for the TXSPIx buffers to empty into the shift register. This is done when the TXS bit (bit 3) of the SPISTATx registers becomes zero.
2. Wait for the SPI shift registers to finish shifting out data. This is done when the SPIF bit (bit 0 of SPISTATx registers) becomes one.
3. Disable the SPI ports by setting the SPIEN bit (bit 0) in the SPICTLx registers to zero.

Stopping DMA Transfers

When performing transmit DMA transfers, data moves through a four deep SPI DMA FIFO, then into the TXSPIx buffers, and finally into the

shift register. DMA interrupts are latched when the I/O processor moves the last word from memory to the peripheral. For the SPI, this means that the SPI "DMA complete" interrupt is latched when there are six words remaining to be transmitted (four in the FIFO, one in the TXSPIx buffers, and one being shifted out of the shift register). To disable the SPI port after a DMA transmit operation, use the following steps:

1. Wait for the DMA FIFO to empty. This is done when the SPISx bits (bits 13–12 in the SPIDMACx registers) become zero.
2. Wait for the TXSPIx registers to empty. This is done when the TXS bit, (bit 3) in the SPISTATx registers becomes zero.

When stopping receive DMA transfers, it is recommended that programs follow the SPI disable steps provided in "Switching from Receive to Receive/Transmit DMA" below.

3. Wait for the SPI shift register to finish transferring the last word. This is done when the SPIF bit (bit 0) of the SPISTATx registers becomes one.
4. Disable the SPI ports by setting theSPIEN bit (bit 0) of theSPICTLx registers to zero.

Switching from Transmit To Transmit/Receive DMA

The following sequence details the steps for switching from transmit to receive DMA.

With disabled SPI:

1. Write 0x00 to the SPICTLx registers to disable SPI. Disabling the SPI also clears the RXSPIx/TXSPIx registers and the buffer status.
2. Disable DMA by writing 0x00 to the SPIDMAxC register.

3. Clear all errors by writing to the W1C-type bits in the SPISTATx registers. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the SPICTLx registers and enable the SPI ports.
5. Configure DMA by writing to the DMA parameter registers and enable DMA.

With enabled SPI:

1. Clear the RXSPIx/TXSPxI registers and the buffer status without disabling the SPI. This can be done by ORing 0xC0000 with the present value in the SPICTLx registers. For example, programs can use the RXFLSH and TXFLSH bits to clear TXSPIx/RXSPIx and the buffer status.
2. Disable DMA by writing 0x00 to the SPIDMAC register.
3. Clear all errors by writing to the W1C-type bits in the SPISTAT register. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the SPICTL register to remove the clear condition on the TXSPI/RXSPI registers.
5. Configure DMA by writing to the DMA parameter registers and enable DMA.

Switching from Receive to Receive/Transmit DMA

Use the following sequence to switch from receive to transmit DMA. Note that TXSPIx and RXSPIx are registers but they may not contain any bits, only address information.

With disabled SPI:

1. Write 0x00 to the SPICTLx registers to disable SPI. Disabling SPI also clears the RXSPIx/TXSPIx register contents and the buffer status.
2. Disable DMA and clear the DMA FIFO by writing 0x80 to the SPIDMACx registers. This ensures that any data from a previous DMA operation is cleared because the SPICLK signal runs for five more word transfers even after the DMA count falls to zero in the receive DMA.
3. Clear all errors by writing to the SPISTATx registers. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the SPICTLx registers and enable SPI.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMACx register.

With enabled SPI:

1. Clear the RXSPIx/TXSPIx registers and the buffer status without disabling the SPI by ORing 0xC0000 with the present value in the SPICTLx registers. Use the RXFLSH (bit 19) and TXFLSH (bit 18) bits in the SPICTLx registers to clear the RXSPIx/TXSPIx registers and the buffer status.
2. Disable DMA and clear the FIFO by writing 0x80 to the SPIDMACx registers. This ensures that any data from a previous DMA operation is cleared because SPICLK runs for five more word transfers even after the DMA count is zero in receive DMA.
3. Clear all errors by writing to the W1C-type bits in the SPISTATx registers. This ensures that no interrupts occur due to errors from a previous DMA operation.

4. Reconfigure the SPICTLx registers to remove the clear condition on the TXSPIx/RXSPIx registers.
5. Configure DMA by writing to the DMA parameter registers (described in "DMA Channel Priority" on page 2-11) and the SPIDMACx registers using the SPIDEN bit (bit 0).

Programming Examples

The example shown in Listing 7-1 transmits a buffer of data from the SPI port in master mode using DMA. In this example, the I/O processor (IOP) automatically moves data from internal memory to the SPI's four-deep DMA FIFO.

The second example, shown in Listing 7-2 on page 7-46, also transmits a buffer, but the transfer is core-driven using interrupts. In this example, only the SPI's one-deep transmit buffer (TXSPI) is serviced by the core and the four-deep DMA FIFO is not used. The core supplies the SPI port with data in a short loop which causes the core to hang at each write to the transmit buffer until the SPI is ready for new data.

The third example, shown in Listing 7-3 on page 7-47, receives multiple buffers using DMA chaining. DMA chaining on the ADSP-2136x processor SPI is initialized differently than on other SHARC processors, as described in Chapter 2, I/O Processor.

Listing 7-1. SPI Master Mode Transmit DMA

```
#include <def21364.h>
#include <sru21364.h>
#include <SRU.h>

.SECTION/DM seg_dmda;
/* Transmit Buffer */
```

```
.var tx_buf[BUFSIZE] =
0x11111111,
0x22222222,
0x33333333,
0x44444444,
0x55555555,
0x66666666,
0x77777777,
0x88888888,
0x99999999,
0xAAAAAAAA;

/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:

/* Init SPI MASTER TX */
r0=0;
dm(SPICTL) = r0;
dm(SPIFLG) = r0;
/* Set up DAG registers */
i4 = tx_buf;
m4 = 1;
ustat3 = DMISO|        /* Disable MISO on transfers */
WL32|                  /* 32-bit words */
SPIMS|                 /* Master mode (internal SPICLK) */
SPIEN|                 /* Enable SPI port */
TIMOD1;                /* Initialize SPI port to begin
                          transmitting when DMA is enabled */
dm(SPICTL) = ustat3;

/* set the SPI baud rate to PCLK/4*64 (650.39KHz @ 167MHz)*/
ustat3 = 0x64;
```

```
dm(SPIBAUD) = ustat3;
/* Set up loop to transmit data */
lcntr = LENGTH(tx_buf), do (pc,4) until lce;

/* Retrieve data using DAG1 and send TX via SPI */
r0 = dm(i4,m4);
dm(TXSPI) = r0;
_main.end: jump (pc,0);
```

Listing 7-2. Core-Driven Interrupt SPI Transfer

```
#define BUFSIZE 0x100
/* Source data to be transmitted via SPI DMA */
.section/dm seg_dmda;
.var src_buf[BUFSIZE] = "source.dat";

/* Application code */
.global _main;
.segment/pm seg_pmco;
_main:

/* Init SPI MASTER TX DMA */
r0 = 0;
dm(SPICTL) = r0;
dm(SPIFLG) = r0;
dm(SPIDMAC) = r0;
r0 = DSOEN;
dm(SPIFLG) = r0;            /*use flag0 as spi device select */
ustat3 = src_buf; dm(IISPI) = ustat3;   /* point to 'src_buf'*/
ustat3 = LENGTH(src_buf); dm(CSPI) = ustat3;   /* count = 256 */
ustat3 = 1; dm(IMSPI) = ustat3;         /* step size = 1 */


/* set the SPI baud rate to PCLK/4*64 (650.39KHz @ 167MHz)*/
ustat3 = 0x64;
dm(SPIBAUD) = ustat3;
```

```
ustat3 = DMISO|             /* Disable MISO on transfers */
WL32|                       /* 32-bit words */
SPIMS|                      /* Master mode (internal SPICLK) */
SPIEN|                      /* Enable SPI port */
TIMOD2;                     /* Initialize SPI port to begin
                               transmitting when DMA is enabled */
dm(SPICTL) = ustat3;
ustat3 = SPIDEN; dm(SPIDMAC) = ustat3;       /* begin DMA */
_main.end: jump (pc,0);
```

Listing 7-3. SPI DMA Chaining Example

```
.section/dm seg_dmda;
/* Destinations for incoming data */
.var dest_bufC[8];
.var dest_bufB[8];
.var dest_bufA[8];

/* Transfer Control Blocks (TCB's) */
.var first_tcb[] =
(0x7FFFF&second_tcb + 3),       /* for CPSPI (next tcb) */
LENGTH(dest_bufB),              /* for CSPI (next count) */
1,                              /* for IMSPI (next modify) */
dest_bufB;                      /* for IISPI (next index) */
.var second_tcb[] = 0,          /* null CPSPI ends chain */
LENGTH(dest_bufC),              /* count for final DMA */
1,                              /* IM for final DMA */
dest_bufC;                      /* II for final DMA */
/* NOTE: Chain Pointer registers must point to the LAST
location in the TCB, "tcb + 3". */

/*Main code section */
.global _main;
.section/pm seg_pmco;
```

```
_main:

/* clear SPI settings */
r0 = 0;
dm(SPICTL) = r0;
dm(SPIFLG) = r0;
dm(SPIDMAC) = r0;

/* setup first DMA in chain */
ustat3 = 8; dm(CSPI) = ustat3;           /* count = 8 words */
ustat3 = 1; dm(IMSPI) = ustat3;          /* step size = 1 */
ustat3 = dest_bufA; dm(IISPI) = ustat3; /* point to dest_bufA */

/* set the SPI baud rate to PCLK/4*64 (650.39KHz @ 167MHz)*/
ustat3 = 0x64;
dm(SPIBAUD) = ustat3;

/* configure processor's SPI slave-select signals */
ustat3 = DSOEN|         /*enable SPI slave device select zero */
SPIFLG3|SPIFLG2|SPIFLG1;     /* Set SPIFLG0 low to */
dm(SPIFLG) = ustat3;         /*select SPI slave on FLAG0 pin */

/* configure SPI port to power-on settings */
ustat3 = CPHASE|      /* sample MISO on second edge of SPICLK */
CLKPL|                /* sampling edge of SPICLK is rising */
WL32|                 /* 32-bit words */
SPIMS|                /* Master mode (internal SPICLK) */
SPIEN|                /* Enable SPI port */
SENDZ|                /* when TXSPI empty, MOSI sends zeros */
TIMOD2;               /* Start SPICLK when DMA is enabled */
dm(SPICTL) = ustat3;

/*configure SPI for chained receive DMA operation */
ustat3 = SPIRCV|       /* DMA direction = receive */
```

```
SPICHEN|                /* enable DMA chaining */
SPIDEN;                 /* enabling DMA initiates the transfer */
dm(SPIDMAC) = ustat3;



/* 1st DMA starts when a valid address is written to CPSPI*/
ustat3 = (0x7FFFF&(first_tcb+3));
dm(CPSPI) = ustat3;           /* point to tcb_A */
_main.end: jump(pc,0);
```

8 INPUT DATA PORT

The Input Data Port (IDP) compromises two units: the serial input port (SIP) and the parallel data acquisition port (PDAP). Located inside the DAI of the SHARC processor, it provides an efficient way of transferring data from DAI pin buffers, the parallel port, the asynchronous sample rate converters (ASRC) and the S/PDIF transceiver to the internal memory of SHARC.

Features

The following list and Table 8-1 describe the IDP features.

Table 8-1. IDP Port Feature Summary

Feature	SIP	PDAP
Connectivity		
Multiplexed Pinout	No	Yes
SRU DAI Required	Yes	Yes
SRU DAI Default Routing	No	No
Interrupt Default Routing	Yes (P0I, P12I)	Yes (P0I, P12I)
Protocol		
Master Capable	No	No
Slave Capable	Yes	Yes
Transmission Simplex	Yes	Yes
Transmission Half Duplex	No	No

Table 8-1. IDP Port Feature Summary (Cont'd)

Feature	SIP	PDAP
Transmission Full Duplex	No	No
Access Type		
Data Buffer	Yes	Yes
Core Data Access	Yes	Yes
DMA Data Access	Yes	Yes
DMA Channels	8	1
DMA Chaining	No	No
Interrupt Source	Core/DMA (DAI)	Core/DMA (DAI)
Boot Capable	No	No
Local Memory	No	No
Clock Operation	PCLK/4	PCLK/4

- Provides a mechanism for a large number of asynchronous channels (up to 8).
- Supports industry standard data formats, I^2S, left-justified and right-justified for serial input ports.
- The PDAP supports four data packing modes for parallel data.
- The PDAP supports a maximum of 20-bits.
- Provides two data transfer types, through DMA or interrupt driven transfer by core.

Functional Description

The IDP has eight serial input ports (SIPs) and one parallel data acquisition port (PDAP). There is a central 8 x 32 FIFO with 8 input channels where data from all the eight SIPs and PDAP are collected. Transfers from this FIFO to internal memory can be performed either via DMA or by interrupts driven by the core.

IDP Channel 0 is shared by SIP0 and PDAP. All other 7 SIPs are connected to corresponding IDP channel of FIFO.

The DMA engine of the IDP implements DMA for all the 8 channels. It has eight sets of DMA parameter registers for 8 channels. Data from channel 0 is directed to internal memory location controlled by set of registers for channel 0 and so on.

The parallel data is acquired through the parallel data acquisition port (PDAP) which provides a means of moving high bandwidth data to the core's memory space. The data may be sent to memory as one 32-bit word per input clock cycle or packed together (for up to four clock cycles worth of data).

Figure 8-1 provides a graphical overview of the input data port architecture. Notice that each channel is independent and contains a separate clock and frame sync input.

The IDP provides an easy way to pump serial data into on-chip memory since it is less complex than the traditional SPORT module, limited to unidirectional slave transfers only.

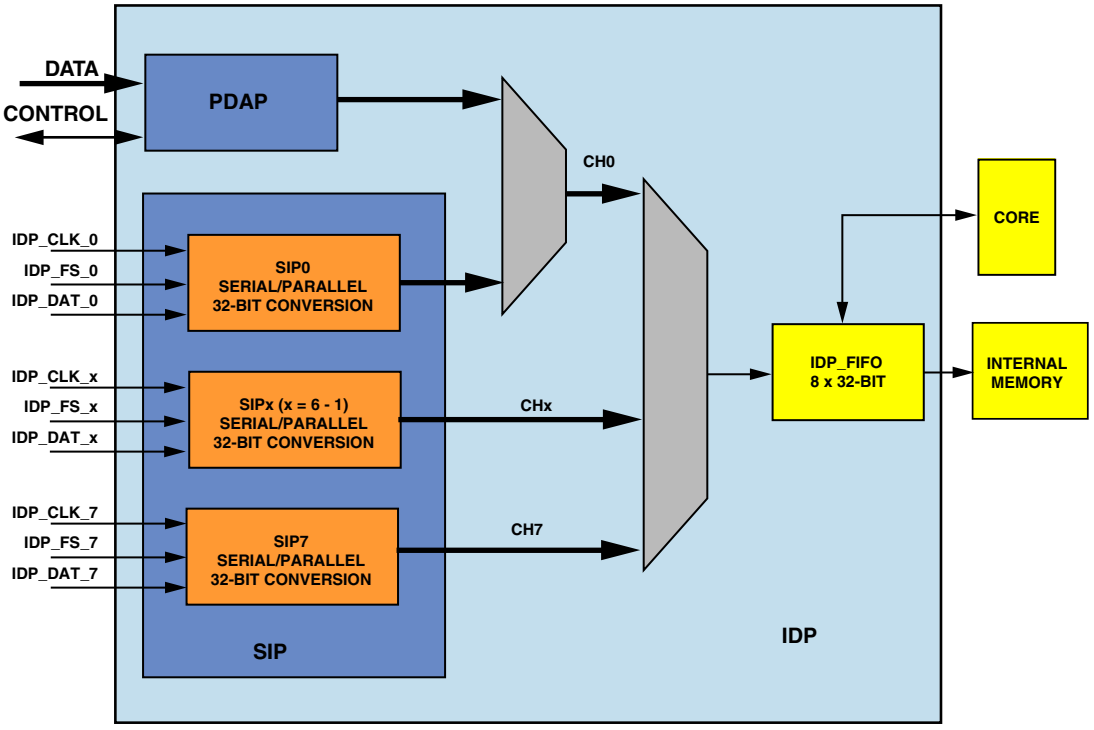


Figure 8-1. Input Data Port

Serial Input Port

The IDP provides up to eight serial input channels—each with its own clock, frame sync, and data inputs. The eight channels are automatically multiplexed into a single 32-bit by eight-deep FIFO. Data is always formatted as a 64-bit frame and divided into two 32-bit words. The serial protocol is designed to receive audio channels in I^2S, left-justified, or right-justified mode. One frame sync cycle indicates one 64-bit left-right pair, but data is sent to the FIFO as 32-bit words (that is, one-half of a frame at a time).

Pin Descriptions

Table 8-2 provides descriptions of the pins used for the serial interface port.

Table 8-2. SIP Pin Descriptions

Internal Node	I/O	Description
IDP7–0_CLK_I	I	**Serial Input Port Receive Clock Input.** This signal must be generated externally and comply to the supported input formats.
IDP7–0_FS_I	I	**Serial Input Port Frame Sync Input.** The frame sync pulse initiates shifting of serial data. This signal must be generated externally and comply to the supported input formats.
IDP7–0_DAT_I	I	**Serial Input Port Data Input.** Unidirectional data pin. Data signal must comply to the supported data formats

SRU Descriptions

The SRU (signal routing unit) needs to be programmed in order to connect the IDP to the output pins as shown in Table 8-3.

Table 8-3. IDP DAI/SRU Signal Connections

Internal Node	DAI Group	SRU Register
Inputs		
IDP7–0_CLK_I	Group A	SRU_CLK3–2
IDP7–0_FS_I	Group C	SRU_FS3–2
IDP7–0_DAT_I	Group B	SRU_DAT5–4

Input Data Formats

An audio signal that is normally 24 bits wide is contained within the 32-bit word. An additional four bits are available for status and formatting data (compliant with the IEC 90958, S/PDIF, and AES3 standards). An additional bit identifies the left-right one-half of the frame. If the data is not in IEC standard format, the serial data can be any data word up to 28 bits wide. The remaining three bits are used to encode one of the eight channels being passed through the FIFO to the core. The FIFO output may feed eight DMA channels, where the appropriate DMA channel (corresponding to the channel number) is selected automatically.

Regardless of mode, bit 3 always specifies whether the data is received in the left channel or the right channel of the corresponding input frame, as shown in Figure 8-2.

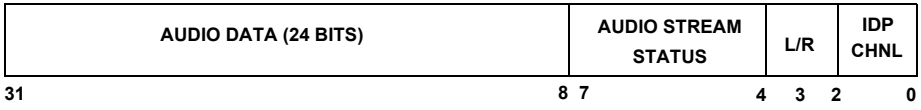


Figure 8-2. Word Format

Note that each input channel has its own clock and frame sync input, so unused IDP channels do not produce data and therefore have no impact on FIFO throughput. The clock and frame sync of any unused input should be routed by the SRU to low to avoid unintentional acquisition. The input data port supports a maximum clock speed of the PCLK/4.

The framing format is selected by using the IDP_SMODEx bits (three bits per channel) in the IDP_CTL0 register. Bits 31–8 of the IDP_CTL0 register control the input format modes for each of the eight channels. The eight groups of three bits indicate the mode of the serial input for each of the eight IDP channels.

Figure 8-3 shows the FIFO data packing for the different serial modes.

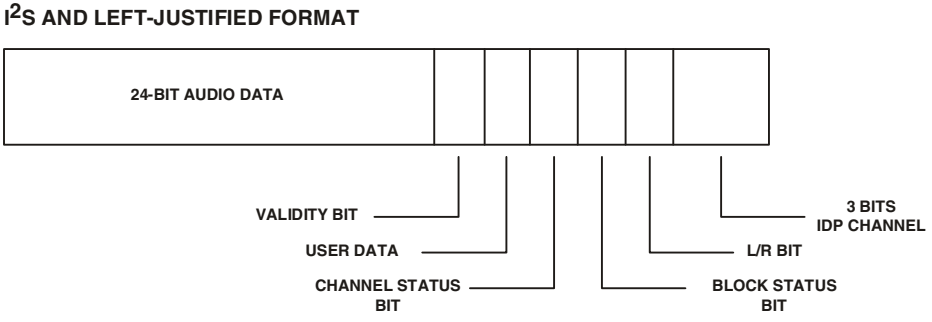


Figure 8-3. FIFO Data Packing

The polarity of left-right encoding isindependent of the serial mode frame sync polarity selected in IDP_SMODE for that channel (Table 8-4). Note that I²S mode uses a low frame sync (left-right) signal to dictate the first (left) channel, and left-justified mode uses a HIGH frame sync (left-right) signal to dictate the first (left) channel of each frame. In either mode, the left channel has bit 3 set (= 1) and the right channel has bit 3 cleared (= 0).

Figure 8-4 shows the relationship between frame sync, serial clock, and left-justified data.

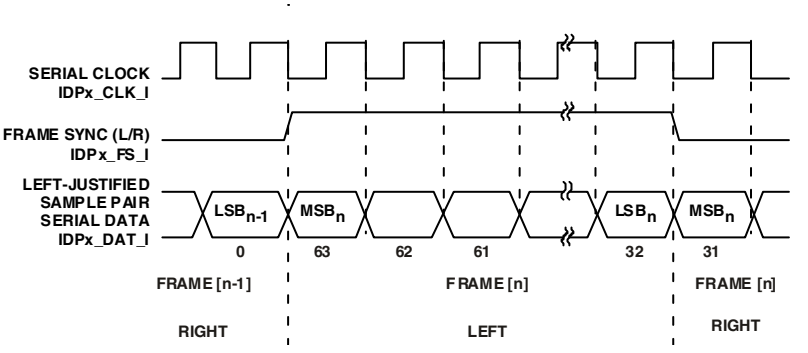


Figure 8-4. Timing in Left-justified Mode

Figure 8-5 shows the relationship between frame sync, serial clock, and I²S data.

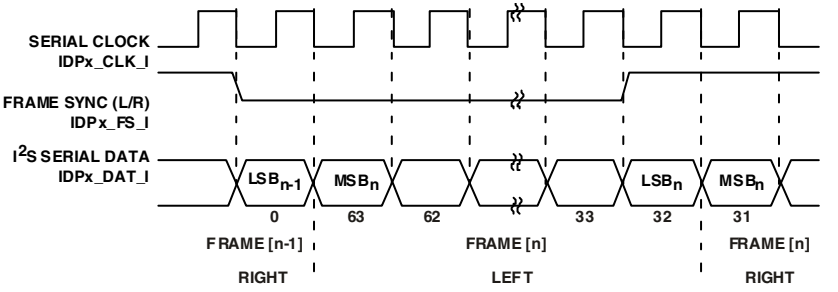


Figure 8-5. Timing in I²S Mode

Register Descriptions

This section provides information on the IDP control and status registers. Complete bit information can be found at "Input Data Port Registers" on page A-47.

Control Registers (IDP_CTLx)

The ADSP-2136x SHARC processors have a new IDP control register IDP_CTL1. The IDP_CTL1-0 registers are used to control the IDP operations. Note the SIP/PDAP modules have a total of 6 different modes enable bits which are required for the different transfer types.

To enable the SIP, two separate bits in two different registers must be set. The first are the global IDP_EN and IDP_DMA_EN bits in the IDP_CTL0 register and the second are the specific channel enable bits which is located in the IDP_CTL1 register.

Status Registers (DAI_STAT0)

Several bits in DAI_STAT0 registers can be used to monitor IDP FIFO operations.

FIFO overflow must be reset manually, using the IDP_CLROVR bit in the IDP_CTL0 register. Writing one to this bit clears the overflow conditions for the channels in the DAI_STAT register. Since IDP_CLROVR is a write-only bit, it always returns low when read.

Parallel Data Acquisition Port (PDAP)

This unit is multiplexed with SIP0. The PDAP provides one clock input, one clock hold input and maximum of 20 parallel data input pins. The positive or negative edge of the clock input is used for data latching. The clock hold input (PDAP_HOLD_I) validates a clock edge—if this input is

high then clock edge is masked for data latching. It supports four types of data packing mode selected by MODE bits in the IDP_PP_CTL register.

Port Selection

The input to channel 0 of the IDP is multiplexed, and may be used either in the serial mode or in a direct parallel input mode. Setting the IDP_PDAP_EN bit high disables the connection of SIP0 to channel 0 of the FIFO. The data inputs can come either from the DAI pins or the external port ADDR pins. This is selected by the IDP_PP_SELECT bit in the IDP_PP_CTL register.

Figure 8-6 shows a block diagram of the PDAP.

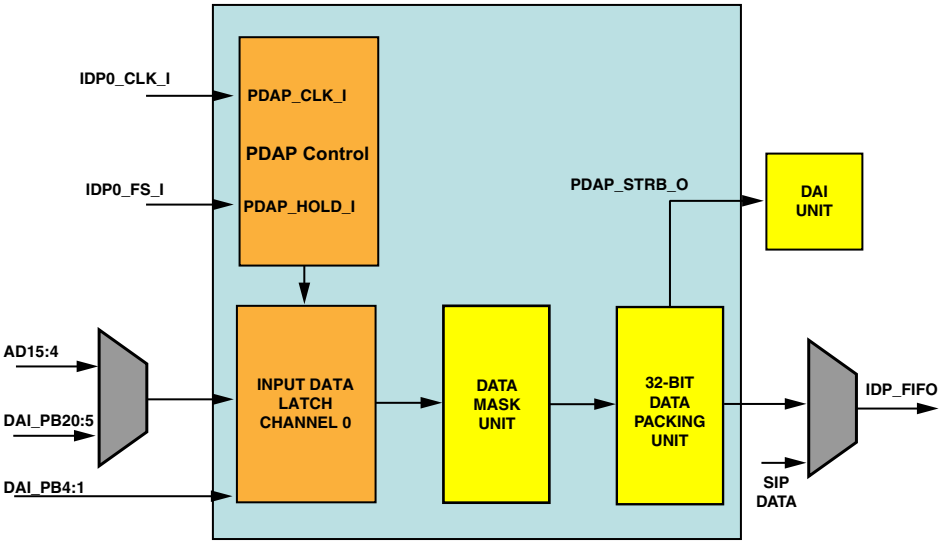


Figure 8-6. PDAP Block Diagram

Pin Descriptions

PDAP signals include 20 data signals and three control signals with one clock (PDAP_CLK_I), one hold (PDAP_HOLD_I), and one PDAP data request strobe (PDAP_STRB_O) signal. The IDP_PP_SELECT (bit 26 of IDP_PP_CTL register) decides the mapping of these signals. For more information, see the port selection bit description (IDP_PP_SELECT) in "Control Register (IDP_PP_CTL)" on page 8-12.

Table 8-4. PDAP Pin Descriptions

Internal Nodes	Type	Description
PDAP_CLK_I	I	**Parallel Data Acquisition Port Clock Input.** Input from the IDP0_CLK_I input. Positive or negative edge of the PDAP clock input is used for data latching depending on the IDP_PDAP_CLKEDGE bit (29) of the IDP_PP_CTL register.
PDAP_HOLD_I	I	**Parallel Data Acquisition Port Frame Sync Input.** The PDAP hold signal determines whether the data is to be latched at an active clock edge or not. When the PDAP hold signal is HIGH, all latching clock edges are ignored and no new data is read from the input pins. The packing unit operates as normal, but it pauses and waits for the PDAP hold signal to be de-asserted and waits for the correct number of distinct input samples before passing the packed data to the IDP FIFO.
PDAP_DATA	I	**Parallel Data Acquisition Port Data Input.** The PDAP latches 20-bit parallel data which where packed into 32-bits by using different packing. Note that input has multiplexed control between the Parallel Port AD15-0 or the DAI_PB20-5 pins.
PDAP_STRB_O	O	**Parallel Data Acquisition Port Clock input.** The PDAP packing unit asserts the output strobe whenever there is 32-bit data available for transfer to the IDP FIFO. The width of this pulse is equal to 2xPCLK cycles. This signal can be used to synchronize external requests for new PDAP data.

SRU Programming

Table 8-5 shows the signal connections when using the PDAP on the DAI pins.

Table 8-5. PDAP DAI/SRU Signal Connections

Internal Node	DAI Connection	SRU Register
Inputs		
PDAP_CLK_I G	roup A	SRU_CLK2
PDAP_HOLD_I	Group C	SRU_FS2
DAI_PB20–1_I	Group D	SRU_PIN4–0
Outputs		
PDAP_STRB_O	Group D, E	

Register Descriptions

This section provides information on the PDAP control register. Complete bit information can be found at "Parallel Data Acquisition Port Control Register (IDP_PP_CTL)" on page A-50.

Control Register (IDP_PP_CTL)

The `IDP_PP_CTL` register (shown in Figure 8-7) are used to control all PDAP operations.

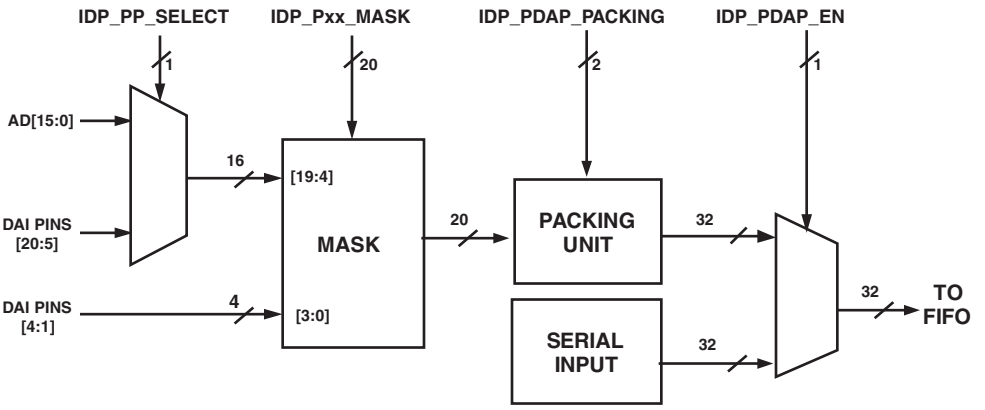


Figure 8-7. Parallel Data Acquisition Port (PDAP) Functions

PDAP Data Packing

Multiple latched parallel sub word samples may be packed into 32-bit words for efficiency. The frame sync input is used to hold off latching of the next sample (that is, ignore the clock edges). The data then flows through the FIFO and is transferredby a dedicated DMA channel into the core's memory as with any IDP channel. As shown in Figure 8-8, the PDAP can accept input words up to 20 bits wide, or can accept input words that are packed as densely as four input words up to eight bits wide.

Mode 11 (No Packing)

Mode 11 provides for 20 bits coming into the packing unit and 32 bits going out to the FIFO in a single cycle. On every clock edge, 20 bits of data are moved and placed in a 32-bit register, left-aligned. That is, bit 19 maps to bit 31. The lower bits, 11–0, are always set to zero, as shown in Figure 8-8.

This mode sends one 32-bit word to FIFO foreach input clock cycle—the DMA transfer rate matches the PDAP input clock rate.

Mode 10 (Packing by 2)

Mode 10 moves data in two cycles. Each input word can be up to 16 bits wide.

- On clock edge 1, bits 19–4 are moved to bits 15–0 (16 bits)
- On clock edge 2, bits 19–4 are moved to bits 31–16 (16 bits)

This mode sends one packed 32-bit word to FIFO for every two input clock cycles—the DMA transfer rate is one-half the PDAP input clock rate.

Mode 01 (Packing by 3)

Mode 01 packs three acquired samples together. Since the resulting 32-bit word is not divisible by three, up to ten bits are acquired on the first clock edge and up to eleven bits are acquired on each of the second and third clock edges:

- On clock edge 1, bits 19–10 are moved to bits 9–0 (10 bits)
- On clock edge 2, bits 19–9 are moved to bits 20–10 (11 bits)
- On clock edge 3, bits 19–9 are moved to bits 31–21 (11 bits)

This mode sends one packed 32-bit word to FIFO for every three input clock cycles—the DMA transfer rate is one-third the PDAP input clock rate.

Mode 00 (Packing by 4)

Mode 00 moves data in four cycles. Each input word can be up to eight bits wide.

- On clock edge 1, bits 19–12 are moved to bits 7–0
- On clock edge 2, bits 19–12 are moved to bits 15–8

- On clock edge 3, bits 19–12 are moved to bits 23–16
- On clock edge 4, bits 19–12 are moved to bits 31–24

This mode sends one packed 32-bit word to FIFO for every four input clock cycles—the DMA transfer rate is one-quarter the PDAP input clock rate.

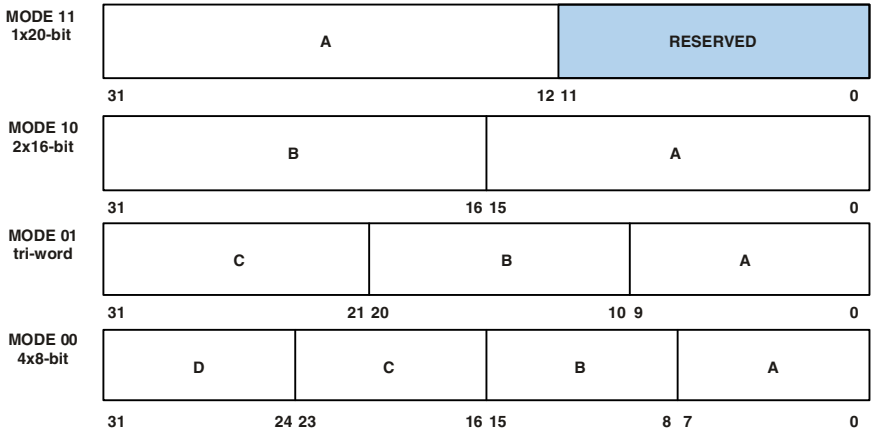


Figure 8-8. Packing Modes in the PDAP

Timing

When the PDAP_HOLD signal is high, all latching clock edges are ignored and no new data is read from the input pins. The packing unit operates as normal, but it pauses and waits for the PDAP_HOLD signal to be de-asserted and waits for the correct number of distinct input samples before passing the packed data to the FIFO.

Figure 8-11 shows packing mode 11 (no packing), Figure 8-10 shows packing mode 10 (packing by 2) and Figure 8-10 shows packing mode 00 (packing by 4).

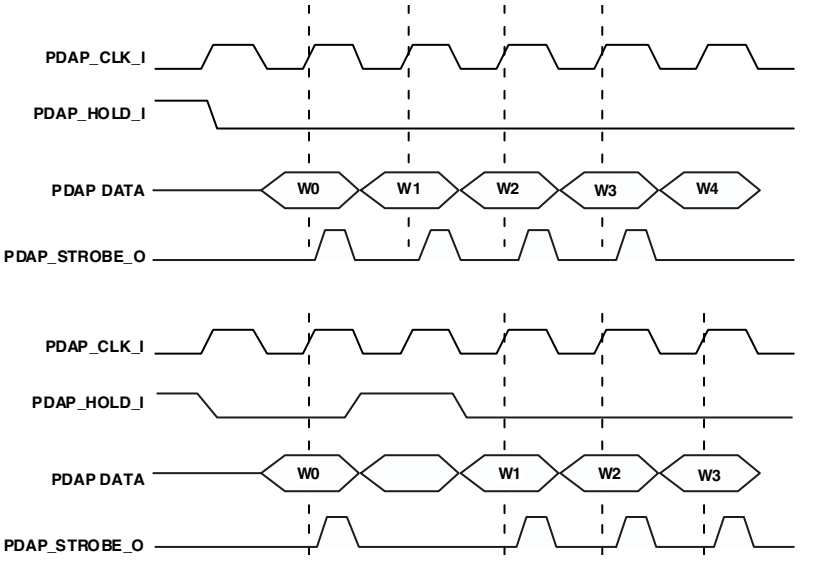


Figure 8-9. PDAP Hold Input (Mode = 11, No Packing)

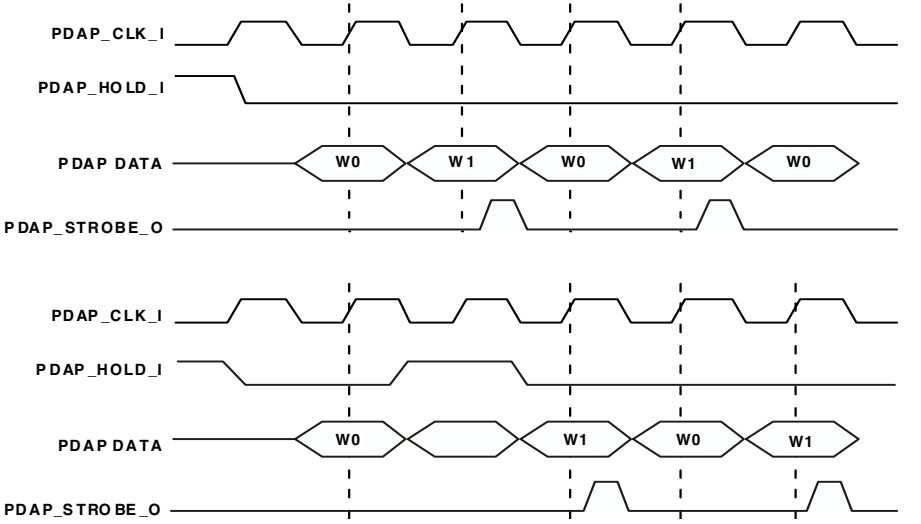


Figure 8-10. PDAP Hold Input (Mode = 10, Pack by 2)

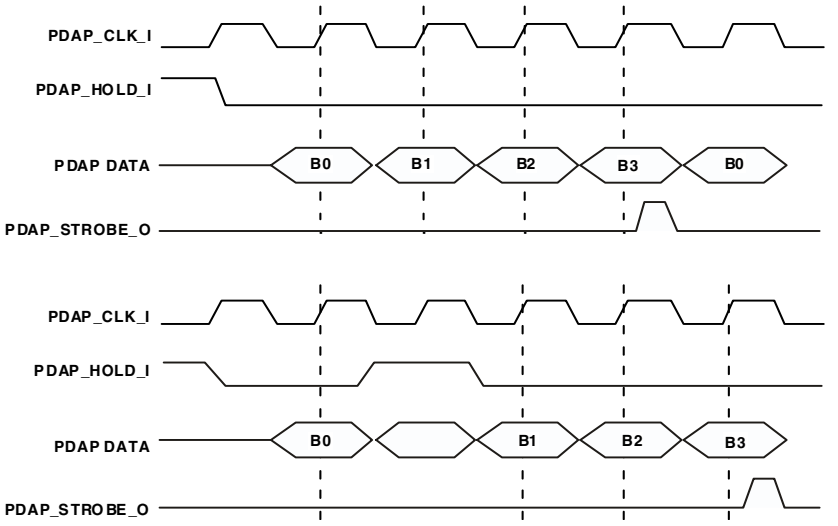


Figure 8-11. PDAP Hold Input (Mode = 00, Pack by 4))

As shown in the figures, PDAP_DATA and PDAP_HOLD are driven by the inactive edges of the clock (falling edge in the above figures) and these signals are sampled by the active edge of the clock (rising edge in the figures).

Data Buffer

The IDP_FIFO register (shown in "Input Data Port FIFO Register (IDP_FIFO)" on page A-53) provides information about the output of the 8-deep IDP FIFO which have been filled by the SIP or the PDAP units. Normally, this register is used only to read and remove the top sample from the FIFO. Channel encoding provides for eight serial input types that correspond to the IDP_SMODEx bits in the IDP control registers. When using channels 0–7 in serial mode, this register format applies. When using channel 0 in parallel mode, refer to the description of the packing bits for PDAP mode.

The information in Table A-22 is not valid when data comes from the PDAP channel.

Data Transfer Types

The data from each of the eight IDP channels is inserted into an eight register deep FIFO, which can only be transferred to the core's memory space sequentially. Data is moved into the FIFO as soon as it is fully received. One of two methods can be used to move data from the IDP FIFO to internal memory:

- The core can remove data from the FIFO manually. This method of moving data from the IDP FIFOis described in the next section, "Core Transfers".
- Eight dedicated DMA channels can sort and transfer data. This method of moving data from the IDP FIFO is described in "DMA Transfers" on page 8-19.

Core Transfers

The core transfers require that the serial peripheral at the SIP writes data to the `IDP_DATAx_I` pin (parallel port or DAI pins for PDAP) according to the selected input format used. These data are automatically moved to the `IDP_FIFO` register without DMA intervention.

The output of the FIFO can be directly fetched by reading from the `IDP_FIFO` buffer. The `IDP_FIFO` buffer is used only to read and remove the top sample from the FIFO, which is a maximum of eight locations deep. When this register is read, the corresponding element is removed from the IDP FIFO, and the next element is moved into the `IDP_FIFO` register. A mechanism is provided to generate an interrupt when more than a specified number of words are in the FIFO. This interrupt signals the core to read the `IDP_FIFO` register.

The number of data samples in the FIFO at any time is reflected in the `IDP_FIFOSZ` bit field (bits 31–28 in the `DAI_STAT0` register), which tracks the number of samples in FIFO.

The three LSBs of FIFO data are the encoded channel number. These are transferred “as is” for this mode. These bits can be used by software to decode the source of data.

The maximum data transfer width to internal memory is 32-bits, as in the case of PDAP data or I^2S and left-justified modes in single channel mode using 32 bits of data. Therefore, PDAP or I^2S and left-justified 32-bit modes cannot be used with other channels in the core/interrupt driven mode since no channel information is available in the data stream.

DMA Transfers

The ADSP-2136x supports two types of DMA transfers, standard and ping-pong.

Eight dedicated DMA channels can sort and transfer the data into one buffer per source channel. When the memory buffer is full, the DMA channel raises an interrupt in the DAI interrupt controller.

DMA Channel Priority

When more than one channel has data ready, the channels always access the `IDP_FIFO` register with fixed priority, from low to high channel number (that is, channel 0 is the highest priority and channel 7 is the lowest priority). For the I/O processor, the eight DMA channels are considered as a group and arbitration can rotate across groups for system balance. For more information, see “DMA Channel Priority” on page 2-11.

Standard DMA

The eight DMA channels each have an index, modify, and count register used for standard DMA. These registers are described below.

- **Internal index registers** (`IDP_DMA_Ix`). Index registers provide an internal memory address, acting as a pointer to the next internal memory location where data is to be written.
- **Internal modify registers** (`IDP_DMA_Mx`). Modify registers provide the signed increment by which the DMA controller post-modifies the corresponding internal memory Index register after each DMA write.
- **Count registers** (`IDP_DMA_Cx`). Count registers indicate the number of words remaining to be transferred to internal memory on the corresponding DMA channel.

A standard DMA access is enabled when the `IDP_EN` bit and `IDP_DMA_EN` bit and the `IDP_DMA_ENx` bits register are set to select a particular channel. The DMA is performed according to the parameters set in the various DMA registers and IDP control registers. An interrupt is generated after end of DMA transfer (when the count = 0).

The `IDP_DMA_ENx` bits (OR the global `IDP_DMA_EN` bit) must be cleared before starting another DMA. An interrupt is generated at the end of DMA transfer.

Ping-Pong DMA

In ping-pong DMA, the parameters have two memory index values (index A and index B), one count value and one modifier value. The DMA starts the transfer with the memory indexed by A. When the transfer is completed as per the value in the count register, the DMA restarts with the memory location indexed by B. The DMA restarts with index A after the transfer to memory with index B is completed as per the count value.

The IDP DMA parameter registers are described below.

- **Internal index registers** (IDP_DMA_Ix, IDP_DMA_IxA, IDP_DMA_IxB). Index A/B registers provide an internal memory address, acting as a pointer to the next internal memory location where data is to be written.
- **Internal modify registers** (IDP_DMA_Mx). Modify registers provide the signed increment by which the DMA controller post-modifies the corresponding internal memory Index register after each DMA write.
- **Ping-Pong Count registers** (IDP_DMA_PCx). Count registers indicate the number of words remaining to be transferred to internal memory on the corresponding DMA channel.

Ping-pong mode is activated when the IDP_EN bit, the IDP_DMA_EN bit, the IDP_DMA_ENx bits, and the IDP_PINGx bits are set for a particular channel. An interrupt is generated after every ping and pong DMA transfer (when the count = 0).

Note that ping-pong DMA is repeated until stopped by resetting the IDP_DMA_ENx bits (OR the global IDP_DMA_EN bit). The IDP_PINGx bits have no affect.

Data Input Format

The LSB bits 2–0 of the data format from the serial inputs are channel encoding bits. Since the data is placed into a separate buffer for each DMA channel (defined by the index registers), these bits are not required and are cleared (=0) when transferring data to internal memory using DMA. However, bit 3 still contains the left/right status information. In the case of PDAP data or 32-bit I^2S and left-justified modes, these three bits are a part of the 32-bit data.

For serial input channels, data is received in an alternating fashion from left and right channels. Data is not pushed into the FIFO as a full left/right frame. Rather, data is transferred as alternating left/right words as it is received. For the PDAP and 32-bit (non-audio) serial input, data is transferred as packed 32-bit words.

Multichannel DMA Operation

The SIP/PDAP can run both standard and ping-pong DMAs in different channels. When running standard DMA, initialize the corresponding IDP_DMA_Ix, IDP_DMA_Mx and IDP_DMA_Cx registers. When running ping-pong DMA, initialize the corresponding IDP_DMA_IxA, IDP_DMA_IxB, IDP_DMA_Mx and IDP_DMA_PCx registers.

DMA transfers for all 8 channels can be interrupted by changing the IDP_DMA_EN bit in the IDP_CTL0 register. None of the other control settings (except for the IDP_EN bit) should be changed. Clearing the IDP_DMA_EN bit (= 0) does not affect the data in the FIFO, it only stops DMA transfers. If the IDP remains enabled, an interrupted DMA can be resumed by setting the IDP_DMA_EN bit again. But resetting the IDP_EN bit flushes the data in the FIFO. If the bit is set again, the FIFO starts accepting new data.

Programs can drop DMA requests from the FIFO if needed. If one channel has finished its DMA, and the global IDP_DMA_EN bit is still set (=1), any data corresponding to that channel is ignored by the DMA machine. This feature is provided to avoid stalling the DMA of other channels, which are still in an active DMA state. To avoid data loss in the finished channel, programs can clear (=0) IDP_DMA_EN bit as discussed in previously.

Multichannel FIFO Status

The state of all eight DMA channels is reflected in the IDP_DMAx_STAT bits (bits 24–17 of DAI_STAT register). These bits are set once the IDP_DMA_EN and IDP_DMA_ENx bits are set, and remain set until the last data from that channel is transferred. Even if IDP_DMA_EN and IDP_DMA_ENx bits remain

set, the IDP_DMAx_STAT bits clear once the required number of data transfers takes place. For more information, see "Digital Applications Interface Status Register (DAI_STAT)" on page A-79.

Note that when a DMA channel is not used (that is, parameter registers are at their default values), the DMA channel's corresponding IDP_DMAx_STAT bit is cleared (= 0).

If the combined data rate from the channels is more than the DMA can service, a FIFO overflow occurs. This condition is reflected for each channel by the individual overflow bits (IDP_FIFO_OVER_CHx) in the DAI_STAT0 register. These are sticky bits that must be cleared by writing to the IDP_CLROVR bit (bit 6 of the IDP_CTL0 register). When an overflow occurs, incoming data from IDP channels is not accepted into the FIFO, and data values are lost. New data is only accepted once space is again created in the FIFO.

Interrupts

This section describes the different types of interrupts.

Core FIFO Threshold Interrupts

When using the interrupt scheme, the IDP_NSET bits (bits 3-0 of the IDP_CTL0 register) can be set to N, so N + 1 data can be read from the FIFO in the interrupt service routine (ISR). The IDP_FIFO_GTN_INT bit in DAI_IRPTL_X register allows to fire flexible interrupts in order to respond with the core under different system conditions.

DMA Interrupts

Using DMA transfer overrides the mechanism used for interrupt-driven core reads from the FIFO. When the IDP_DMA_EN bit and at least one IDP_DMA_ENx of the IDP_CTL1 register are set, the eighth interrupt (IDP_FIFO_GTN_INT) in the DAI_IRPTL_x registers is NOT generated.

At the end of the DMA transfer for individual channels, interrupts are generated. These interrupts are generated after the last DMA data from a particular channel has been transferred to memory. These interrupts (IDP_DMAx_INT) are mapped from the bits 17–10 in the DAI_IRPTL_x registers and generate interrupts when they are set (= 1). These bits are ORed and reflected in high level interrupts that are sent to the DAI interrupt.

An interrupt is generated at the end of a DMA, which is cleared by reading the DAI_IRPTL_x registers.

FIFO Overflow Interrupts

If the data out of the FIFO (either through DMA or core reads) is not sufficient to transfer at the combined data rate of all the channels, then a FIFO overflow can occur. When this happens, new data is not accepted. Additionally, data coming from the serial input channels (except for 32-bit I^2S and left-justified modes) are not accepted in pairs, so that alternate data from a channel is always from left and right channels. If overflow occurs, an interrupt is generated if the IDP_FIFO_OVR_INT bit in the DAI_IRPTL_x register is set (sticky bits in DAI_STAT0 register are also set). Data is accepted again when space has been created in the FIFO.

Note that the total FIFO depth per channel is 9 locations: 1 location for SIP to parallel data conversion + 8 locations for the IDP_FIFO.

Servicing Interrupts

The DAI interrupt controller contains different levels of interrupts. For servicing DAI interrupts refer to "Interrupt Controller" on page 5-24.

Debug Features

The following sections describe the features available for debugging the IDP.

Buffer Hang Disable

The IDP_BHD bit in IDP_CTL0 is used for buffer hang disable control. When there is no data in the FIFO, reading the IDP_FIFO register causes the core to hang. This condition continues until the FIFO contains valid data. Setting the IDP_BHD bit (= 1) prevents the core from hanging on reads from an empty IDP_FIFO register. Clearing this bit (= 0) causes the core to hang under the conditions described previously.

If the IDP_BHD bit (bit 4 in the IDP_CTL0 register) is not set, attempts to read more data than is available in the FIFO results in a core hang.

Shadow Registers

The DAI interrupt controller contains shadow registers to simplify debug techniques since these register are not updated. A read of the DAI_IRPTL_x_SH register provides the same data as a read of the DAI_IRPTL_x register. Reading these DAI shadow registers (DAI_IRPTL_x_SH) does not destroy the contents of the DAI_IRPTL_x registers. For more information, refer to "Interrupt Controller" on page 5-24.

Core FIFO Write

The core may also write to the FIFO.When it does, the audio data word is pushed into the input side of the FIFO (as if it had come from the SRU on the channel encoded in the three LSBs). This can be useful for verifying the operation of the FIFO, the DMA channels, and the status portions of the IDP. The IDP_STAT1 register returns the current state of the read/write index pointers from FIFO.

IDP Effect Latency

The IDP is ready to start receiving data one serial clock cycle (SCLK) after it is enabled by setting IDP_EN bit. No LRCLK edges are lost from this point on.

Disabling IDP DMA by resetting the IDP_DMA_EN bit requires 1 PCLK cycle. Disabling an individual DMA channel by resetting the IDP_DMA_ENx bit requires 2 PCLK cycles.

Programming Model

The following sections provide procedures that are helpful when programming the input data port.

Setting Miscellaneous Bits

This sequence is used in most following programming models as intermediate step.

Set the required values for:

- IDP_SMODEx bits in the IDP_CTLx register to specify the frame sync format for the serial inputs (left-justified I^{2}S, or right-justified mode).
- IDP_Pxx_PDAPMASK bits in the IDP_PP_CTL register to specify the input mask, if the PDAP is used.
- IDP_PP_SELECT bits in the IDP_PP_CTL register to specify input from the DAI pins or the AD15-0 pins, if the PDAP is used.
- IDP_PDAP_CLKEDGE bit (bit 29) in the IDP_PP_CTL register to specify if data is latched on the rising or falling clock edge, if the PDAP is used.

Starting Core Interrupt-Driven Transfer

To start a core interrupt-driven data transfer:

1. Clear the FIFO by setting (= 1) IDP_FFCLR bit (bit 31 in the IDP_CTL1 register).
2. Keep the clock and frame sync inputs of all serial inputs and/or the PDAP connected to low by setting proper values in the SRU registers.
3. Refer to "Setting Miscellaneous Bits" above.
4. Set the IDP_FIFO_GTN_INT bit (bit 8 of the DAI_IRPTL_RE register) to HIGH and set the corresponding bit in the DAI_IRPTL_FE register to low to unmask the interrupt. Set bit 8 of the DAI_IRPTL_PRI register (IDP_FIFO_GTN_INT) as needed to generate a high priority or low priority core interrupt when the number of words in the FIFO is greater than the value of *N* set.
5. Enable the PDAP by setting IDP_PDAP_EN (bit 31 in the IDP_PP_CTL register), if required.
6. Enable the IDP by setting the IDP_EN bit (bit 7 in the IDP_CTL0 register) and the IDP_ENx bits in the IDP_CTL1 register.

Starting A Standard DMA Transfer

To start a DMA transfer from the FIFO to memory:

1. Clear the FIFO by setting (= 1) the IDP_FFCLR bit (bit 31 in the IDP_CTL1 register).
2. While the global IDP_DMA_EN and the IDP_EN bits are cleared (= 0), set the values for the DMA parameter registers that correspond to channels 7–0.

3. Keep the clock and the frame sync input of the serial inputs and/or the PDAP connected to low, by setting proper values in the SRU registers.
4. Refer to "Setting Miscellaneous Bits" above.
5. Enable the channel's IDP_ENx and IDP_DMA_ENx bit settings.
6. Rout all of the required inputs to the IDP by writing to the SRU registers
7. Start the DMA by setting
    - The IDP_PDAP_EN bit (bit 31 in IDP_PP_CTL register if the PDAP is required).
    - The global IDP_DMA_EN bit of the IDP_CTL1 register to enable standard DMA on the selected channel.
    - The global IDP_EN bit (bit 7 in the IDP_CTL0 register).

Starting a Ping-Pong DMA Transfer

To start a ping-pong DMA transfer from the FIFO to memory:

1. Clear the FIFO by setting (= 1) the IDP_FFCLR bit (bit 31 in the IDP_CTL1 register).
2. While the global IDP_DMA_EN and IDP_EN bits are cleared (=0), set the values for the following DMA parameter registers that correspond to channels 7–0.
3. Keep the clock and the frame sync input of the serial inputs and/or the PDAP connected to low, by setting proper values in the SRU registers.
4. Refer to "Setting Miscellaneous Bits" above.

5. Connect all of the inputs to the IDP by writing to the SRU registers.
6. Enable the channel's IDP_ENx, IDP_DMA_ENx and IDP_PINGx bit settings.
7. Start DMA by setting:
    - The IDP_PDAP_EN bit (bit 31 in IDP_PP_CTL register if the PDAP is required).
    - The global IDP_DMA_EN bit of the IDP_CTL1 register to enable the standard DMA of the selected channel.
    - The global IDP_EN bit (bit 7 in the IDP_CTL0 register).

Servicing Interrupts for DMA

The following steps describe how to handle an IDP ISR for DMA.

1. An interrupt is generated and program control jumps to the ISR when the DMA for a channel completes.
2. The program clears the IDP_DMA_EN bit in the IDP_CTL register.
    a. To ensure that the DMA of a particular IDP channel is complete, (all data is transferred into internal memory) wait until the IDP_DMAx_STAT bit of that channel becomes zero in the DAI_STAT register. This is required if a high priority DMA (for example a SPORT DMA) is occurring at the same time as the IDP DMA.
    b. As each DMA channel completes, a corresponding bit in either the DAI_IRPTL_L or DAI_IRPTL_H register for each DMA channel is set (IDP_DMAx_INT). Refer to "DAI Interrupt Controller Registers" on page A-77 for more information on the DAI_IRPTL_L or DAI_IRPTL_H registers.

3. The program clears (= 0) the channel's IDP_DMA_ENx bit in the IDP_CTL1 register which has finished.
4. Reprogram the DMA registers for finished DMA channels.

   More than one DMA channel may have completed during this time period. For each channel, a bit is latched in the DAI_IRPTL_L or DAI_IRPTL_H registers. Ensure that the DMA registers are reprogrammed. If any of the channels are not used, then their clock and frame syncs should be held low.
5. Read the DAI_IRPTL_L or DAI_IRPTL_H registers to see if more interrupts have been generated.
   - If the value(s) are not zero, repeat step 4.
   - If the value(s) are zero, continue to step 6.
6. Re-enable the IDP_DMA_EN bit in the IDP_CTL register (set to 1).
7. Exit the ISR.

If a zero is read in step 5 (no more interrupts are latched), then all of the interrupts needed for that ISR have been serviced. If another DMA completes after step 5 (that is, during steps 6 or 7), as soon as the ISR completes, the ISR is called again because the OR of the latched bits will not be nonzero again. DMAs in progress run to completion.

If step 5 is not performed, and a DMA channel expires during step 4, then, when IDP DMA is re-enabled, (step 6) the completed DMA is *not* reprogrammed and its buffer overruns.

Programming Example

Listing 8-1 shows a data transfer using an interrupt service routine (ISR). The transfer takes place through the digital applications interface (DAI). This code implements the algorithm outlined in "Data Transfer Types" on page 8-18.

Listing 8-1. Interrupt-Driven Data Transfer

```
#include <def21364.h>
#include <sru21364.h>
#include <SRU.h>

/* Using Interrupt-Driven Transfers from the IDP FIFO */
.section/dm seg_dmda;
.var OutBuffer[6];

.section/pm seg_pmco;

initIDP:
r0 = IDP_CLROVR;          /* Reset the IDP FIFO*/
dm(IDP_CTL) = r0;

r0 = BCLR r0 BY 10;       /* Set IDP serial input channel 0 */
r0 = BCLR r0 BY 9;        /* to receive in I2S format */
dm(IDP_CTL) = r0;

/* Connect the clock, data and frame sync of IDP */
/* channel 0 to DAI pin buffers 10, 11 and 12. */
/* Connect IDP0_CLK_I to DAI_PB10_O */
/* Connect IDP0_DAT_I to DAI_PB11_O */
/* Connect IDP0_FS_I to DAI_PB12_O */

SRU(DAI_PB10_O, IDP0_CLK_I);
```

```
SRU(DAI_PB11_O, IDP0_DAT_I);
SRU(DAI_PB12_O, IDP0_FS_I);

/* Pin buffers 10, 11 and 12 are always being used as */
/* inputs. Tie their enables to LOW (never driven). */
/* Connect PBEN10_I to LOW */
/* Connect PBEN11_I to LOW */
/* Connect PBEN12_I to LOW */

SRU(LOW, PBEN10_I);
SRU(LOW, PBEN11_I);
SRU(LOW, PBEN12_I);

/* Assign a value to N_SET. An interrupt will be raised */
/* when there are N_SET+1 words in the FIFO. */
r0 = dm(IDP_CTL);            /* N_SET = 6 */
r0 = BCLR r0 BY 0;
r0 = BSET r0 BY 1;
r0 = BSET r0 BY 2;
r0 = BCLR r0 BY 3;
dm(IDP_CTL) = r0;

ustat1 = dm(DAI_IRPTL_RE);        /* Unmask for rising edge */
bit set ustat1 IDP_FIFO_GTN_INT;
dm(DAI_IRPTL_RE) = ustat1;
ustat1 = dm(DAI_IRPTL_FE);         /* Mask for falling edge */
bit set ustat1 IDP_FIFO_GTN_INT;
dm(DAI_IRPTL_FE) = ustat1;
ustat1 = dm(DAI_IRPTL_PRI);  /* Map to high priority in core */
bit set ustat1 IDP_FIFO_GTN_INT;
dm(DAI_IRPTL_PRI) = ustat1;
ustat1 = dm(IDP_CTL);              /* Start the IDP */
bit set ustat1 IDP_EN;
dm(IDP_CTL) = ustat1;
```

```
initIDP.end:

IDP_ISR:
i0 = OutBuffer;
m0 = 1;
l0 = 0;
LCNTR = 7, DO RemovedFromFIFO UNTIL LCE;
r0 = dm(IDP_FIFO);
dm(i0,m0) = r0;
RemovedFromFIFO:
RTI;
IDP_ISR.end:
```

9 PERIPHERAL TIMERS

In addition to the internal core timer, (using the TMREXP pin as output), the ADSP-2136x processor processors contain identical 32-bit peripheral timers that can be used to interface with external devices. Each timer can be individually configured in three operation modes.

Features

The peripheral timers have the features shown in Table 9-1. The timer block diagram is shown in Figure 9-1.

Table 9-1. Timers Feature Summary

Feature	Timer2–0
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	Yes
Interrupt Default Routing	Yes (P2I, P10I, P17I)
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A

Table 9-1. Timers Feature Summary (Cont'd)

Feature	Timer2–0
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Interrupt Source	Core
Boot Capable	N/A
Local Memory	No
Clock Operation	PCLK

- Independent general-purpose timers
- Three operation modes (PWM, Width capture, external watchdog)
- Global control/status registers for synchronous operation of multiple timers
- Buffered timer registers (Period and Width) to allow changes on the fly

The core timer is controlled by system registers while the peripheral timers are controlled by memory-mapped registers.

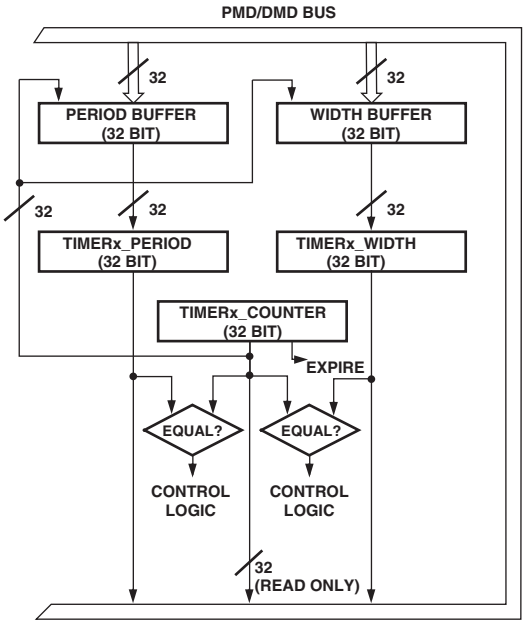


Figure 9-1. Timer Block Diagram

Pin Descriptions

The timer has only one pin which acts as input or output based on the timer mode as shown in Table 9-2.

Table 9-2. Peripheral Timer Pin Descriptions

Internal Node	Type	Description
TIMER2–0_I	I	**Timer Signal.** This input is active sampled during pulse width and period capture (width capture mode) or external event watchdog (external clock mode).

Table 9-2. Peripheral Timer Pin Descriptions

Internal Node	Type	Description
TIMER2–0_O	O	**Timer Signal.** This output is active driven in pulse width modulation (PWM out mode).
TIMER2–0_PBEN_O	O	**Timer Pin Buffer Enable Output Signal.** This output is only driven in PWM out mode.

SRU Programming

Since the timer has operation modes for input (capture and external clock mode) and output (PWM out mode), it requires bidirectional junctions. Table 9-3 shows the required SRU routing. See also "Routing Capabilities" on page 5-20.

Table 9-3. Timer DAI/SRU Signal Connections

Internal Node	DAI Group	SRU Register
Inputs		
TIMER2–0_I	Group E	SRU_EXT_MISCB
Outputs		
TIMER2–0_O	Group D, E	
TIMER2–0_PBEN_O	Group F	

Functional Description

Each timer (Figure 9-2 on page 9-10) has one dedicated bidirectional chip signal, `TIMERx`. The two timer signals are connected to the 20 digital application interface (DAI) pins through the signal routing unit (SRU). The timer signal functions as an output signal in PWM_OUT mode and as an input signal in WDTH_CAP and EXT_CLK modes. To provide these

functions, each timer has four, 32-bit registers. The registers for each timer are:

- Timer x control (TMxCTL) register
- Timer x word count (TMxCNT) register
- Timer x word period (TMxPRD) register
- Timer x word pulse width (TMxW) register

The timers also share a common status and control register—the timer global status and control (TMSTAT) register.

For information on the timer registers, see "Peripheral Timer Registers" on page A-56.

When clocked internally, the clock source is the ADSP-2136x processor's peripheral clock (PCLK). The timer produces a waveform with a period equal to 2 x TMxPRD and a width equal to 2 x TMxW. The period and width are set through the TMxPRD30-0 and the TMxW30-0 bits. Bit 31 is ignored for both.

Register Descriptions

The following sections provide brief descriptions of the primary registers used to program the timers. For more information, see "Peripheral Timer Registers" on page A-56.

Count Registers

The following registers program how the timers operate in the three modes.

Counter Registers (TMxCNT)

When disabled, the timer counter retains its state. When re-enabled, the timer counter is re initialized from the period/width registers based on configuration and mode. The timer counter value should not be set directly by the software. It can be set indirectly by initializing the period or width values in the appropriate mode. The counter should only be read when the respective timer is disabled. This prevents erroneous data from being returned.

Period Registers (TMxPRD)

When enabled and running, the processor writes new values to the timer period and pulse width registers. The writes are buffered and do not update the registers until the end of the current period (when the timer counter register equals the timer period register).

During the *pulse width modulation* (PWM_OUT), the period value is written into the timer period registers. Both period and width register values must be updated "on the fly" since the period and width (duty cycle) change simultaneously. To insure the period and width value concurrency, a 32-bit period buffer and a 32-bit width buffer are used.

During the *pulse width and period capture* (WDTH_CAP) mode, the period values are captured at the appropriate time. Since both the period and width registers are read-only in this mode, the existing 32-bit period and width buffers are used.

During the *external event watchdog* (EXT_CLK) mode, the period register is write-only. Therefore, the period buffer is used in this mode to insure high/low period value coherency.

Pulse Width Register (TMxW)

During the pulse width modulation (PWM_OUT), the width value is written into the timer width registers. Both width and period register val-

ues must be updated “on the fly” since the period and width (duty cycle) change simultaneously. To insure period and width value concurrency, a 32-bit period buffer and a 32-bit width buffer are used.

During the pulse width and period capture (WDTH_CAP) mode, both the period and width values are captured at the appropriate time. Since both the width and period registers are read-only in this mode, the existing 32-bit period and width buffers are used.

When the processor is in EXT_CLK mode, the width register is unused.

Status and Control Registers

The timer global status and control (TMSTAT) register indicates the status of all three timers using a single read. The TMSTAT register also contains timer enable bits. Within TMSTAT, each timer has a pair of sticky status bits, that require a write one-to-set (TIMxEN) or write one-to-clear (TIMxDIS) to enable and disable the timer respectively.

Writing a one to both bits of a pair disables that timer.

Each timer also has an overflow error detection bit, TIMxOVF. When an overflow error occurs, this bit is set in the TMSTAT register. A program must write one-to-clear this bit.

After the timer has been enabled, its TIMxEN bit is set (= 1). The timer then starts counting three peripheral clock cycles (PCLK) after the TIMxEN bit is set. Setting (writing one to) the timer’s TIMxDIS bit stops the timer without waiting for another event.

Operation

To enable an individual timer, set the timer’s TIMxEN bit in the TMSTAT register. To disable an individual timer, set the timer’s TIMxDIS bit in the

TMSTAT register. To enable all three timers in parallel, set all the TIMxEN bits in the TMSTAT register.

Before enabling a timer, always program the corresponding timer's configuration (TMxCTL) register. This register defines the timer's operating mode, the polarity of the TIMERx signal, and the timer's interrupt behavior. Do not alter the operating mode while the timer is running. For more information on the TMxCTL register, see "Timer Configuration Registers (TMxCTL)" on page A-56.

Mode Selection

The three operating modes of the peripheral timer; PWM_OUT, WDTH_CAP, and EXT_CLK, are described in Table 9-4 and the following sections.

Table 9-4. Timer Signal Use

TMxCTL Register Settings	PWM_OUT Mode	WIDTH_CAP Mode	EXT_CLK Mode
MODE	01 = Output PWM Waveform	10 = Input Waveform	11 = Input Event
TIMEN	1 = Enable & Start Timer 0 = Disable Timer	1 = Enable & Start Timer 0 = Disable Timer	1 = Enable & Start Timer 0 = Disable Timer
PULSE	1 = Generate High Width 0 = Generate Low Width	1 = Measure High Width 0 = Measure Low Width	Count at event rise only
PRDCNT	1 = Generate PWM 0 = Single Width Pulse	1 = Measure Period 0 = Measure Width	Unused
IRQEN	1 = Enable Interrupt 0 = Disable Interrupt	1 = Enable Interrupt 0 = Disable Interrupt	1 = Enable Interrupt 0 = Disable Interrupt
Period	WO: Period Value	RO: Period Value	WO: Period Value
Width	WO: Width Value	RO: Width Value	Unused
Counter	RO: Only if not enabled Counts down on PCLK	RO: Only if not enabled Counts up on PCLK	RO: Only if not enabled Counts down on Event

Table 9-4. Timer Signal Use (Cont'd)

TMxCTL Register Settings	PWM_OUT Mode	WIDTH_CAP Mode	EXT_CLK Mode
OVF_ERR (IRQ also set)	Set if Initialized with: Period < Width or Period == Width or Period == 0	Set if the Counter wraps (Error Condition)	Unused
IRQ (If enabled)	If PERIOD_CNT: 1 = Set at end of Period 0 = Set at end of Width	If PERIOD_CNT: 1 = Set at end of Period 0 = Set at end of Width	Set after Period Expires and PCLK is running

Pulse Width Modulation Mode (PWM_OUT)

In PWM_OUT mode, the timer supports on-the-fly updates of period and width values of the PWM waveform. The period and width values can be updated once every PWM waveform cycle, either within or across PWM cycle boundaries.

To enable PWM_OUT mode, set the TIMODE1-0 bits to 01 in the timer's configuration (TMxCTL) register. This configures the timer's TIMERx signal as an output with its polarity determined by PULSE as follows:

- If PULSE is set (= 1), an active high width pulse waveform is generated at the TIMERx signal.
- If PULSE is cleared (= 0), an active low width pulse waveform is generated at the TIMERx signal.

The timer is actively driven as long as the TIMODE field remains 01.

Figure 9-2 shows a flow diagram for PWM_OUT mode. When the timer becomes enabled, the timer checks the period and width values for plausi-

bility (independent of the value set with the PRDCNT bit) and does *not* start to count when any of the following conditions are true:

- Width is equal to zero
- Period value is lower than width value
- Width is equal to period

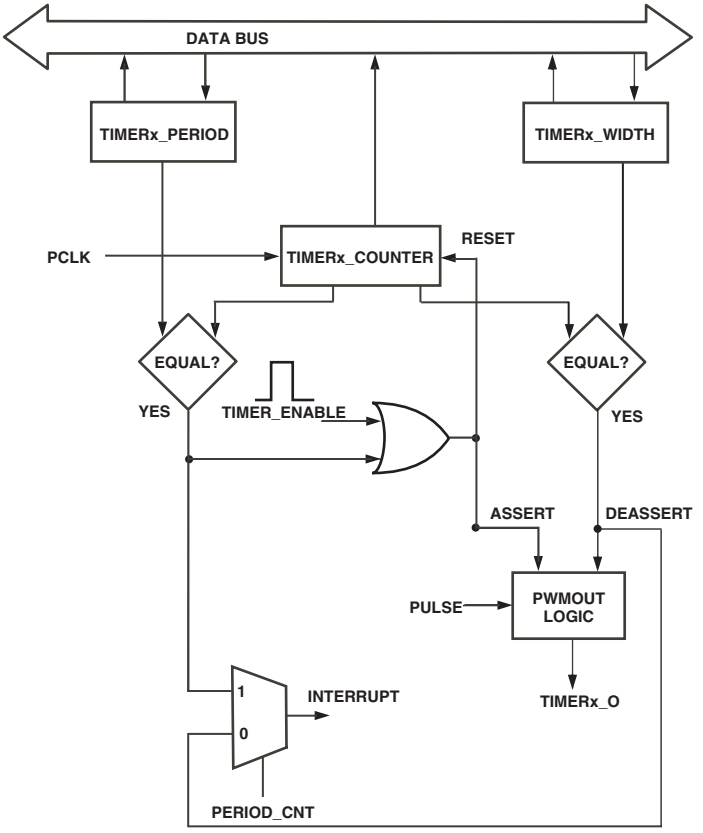


Figure 9-2. Timer Flow Diagram – PWM_OUT Mode

On invalid conditions, the timer sets both the TIMxOVF and the TIMIRQx bits and the Count register is not altered. Note that after reset, the timer registers are all zero. The PWM_OUT timing is shown in Figure 9-3.

As mentioned earlier, 2 x TMxPRD is the period of the PWM waveform and 2 x TMxW is the width. If the period and width values are valid after the timer is enabled, the count register is loaded with the value resulting from 0xFFFF FFFF – width. The timer counts upward to 0xFFFF FFFF. Instead of incrementing to 0xFFFF FFFF, the timer then reloads the counter with the value derived from 0xFFFF FFFF – (period – width) and repeats.

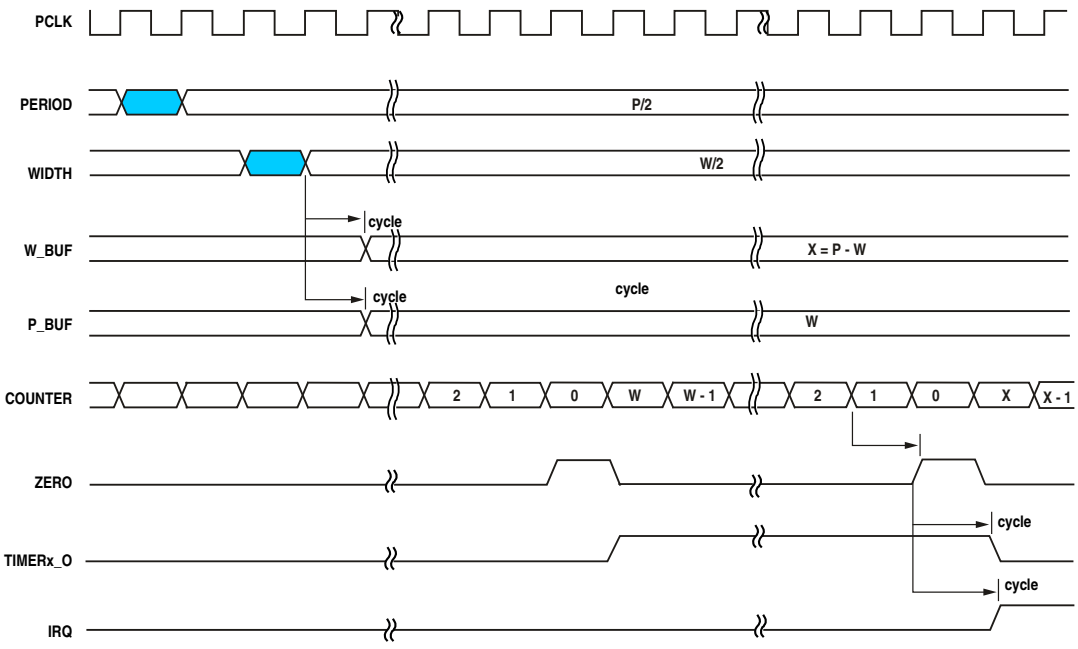


Figure 9-3. PWM_OUT Timing

PWM Waveform Generation

If the PRDCNT bit is set, the internally-clocked timer generates rectangular signals with well-defined period and duty cycles. This mode also generates periodic interrupts for real-time processing.

The 32-bit period (TMxPRD) and width (TMxW) registers are programmed with the values of the timer count period and pulse width modulated output pulse width.

When the timer is enabled in this mode, the TIMERx signal is pulled to a deasserted state each time the pulse width expires, and the signal is asserted again when the period expires (or when the timer is started).

To control the assertion sense of the TIMERx_O signal, the PULSE bit in the corresponding TMxCTL register is either cleared (causes a low assertion level) or set (causes a high assertion level).

When enabled, a timer interrupt is generated at the end of each period. An ISR must clear the interrupt latch bit TIMxIRQ and might alter period and/or width values. In pulse width modulation applications, the program needs to update the period and pulse width values while the timer is running.

When a program updates the timer configuration, the TMxW register must always be written to last, even if it is necessary to update only one of the registers. When the TMxW value is not subject to change, the ISR reads the current value of the TMxW register and rewrite it again. On the next counter reload, all of the timer control registers are read by the timer.

To generate the maximum frequency on the TIMERx_O output signal, set the period value to two and the pulse width to one. This makes the TIMERx signal toggle every 2 PCLK clock cycles as shown in Figure 9-9. Assuming PCLK = 133 MHz:

Maximum period = $2 \times (2^{31} - 1) \times 7.5$ ns = 32 seconds.

If your application requires a more sophisticated PWM output generator, refer to Chapter 10, Pulse Width Modulation.

Single-Pulse Generation

If the PRDCNT bit is cleared, the PWM_OUT mode generates a single pulse on the TIMERx_O signal. This mode can also be used to implement a well defined software delay that is often required by state machines. The pulse width (= 2 x TMxW) is defined by the width register and the period register should be set to a value greater than the pulse width register.

At the end of the pulse, the interrupt latch bit (TIMxIRQ) is set and the timer is stopped automatically. If the PULSE bit is set, an active high pulse is generated on the TIMERx_O signal. If the PULSE bit is not set, the pulse is active low.

Pulse Mode

The waveform produced in PWM_OUT mode with PRDCNT = 1 normally has a fixed assertion time and a programmable deassertion time (via the TMxW register). When three timers are running synchronously by the same period settings, the pulses are aligned to the asserting edge as shown in Figure 9-4.

Note that the timer does not support toggling of the PULSE bit in each period.

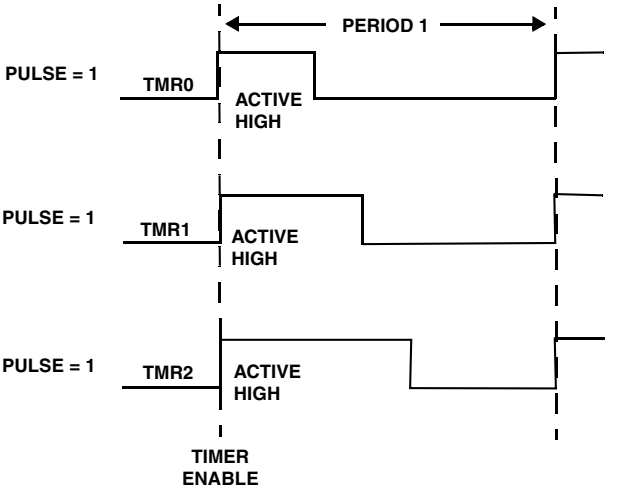


Figure 9-4. Timers with Pulses Aligned to Asserting Edge

Pulse Width Count and Capture Mode (WDTH_CAP)

To enable WDTH_CAP mode, set the TIMODE1-0 bits in the TMxCTL register to 10. This configures the TIMERx signal as an input signal with its polarity determined by PULSE. If PULSE is set (= 1), an active high width pulse waveform is measured at the TIMER_Ix signal. If PULSE is cleared (= 0), an active low width pulse waveform is measured at theTIMERx_I signal. The internally-clocked timer is used to determine the period and pulse width of externally-applied rectangular waveforms. The period and width registers are read-only in WDTH_CAPmode. The period and pulse width measurements are with respect to a clock frequency of PCLK ÷ 2.

Figure 9-5 shows a flow diagram for WDTH_CAP mode. In this mode, the timer resets words of the count in the TMxCNT register value to 0x0000 0000 and does not start counting until it detects the leading edge on the TIMERx_I signal.

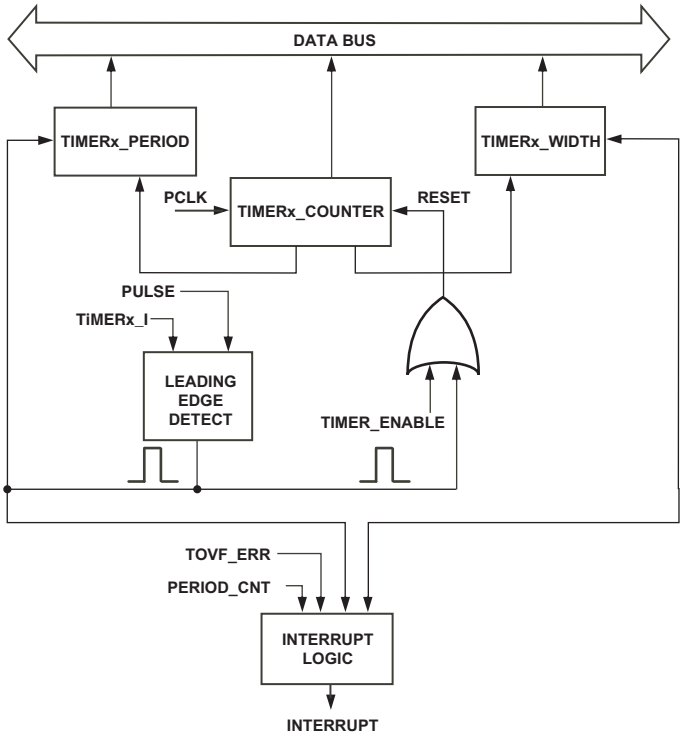


Figure 9-5. Timer Flow Diagram – WDTH_CAP Mode

When the timer detects a first leading edge, it starts incrementing. When it detects the trailing edge of a waveform, the timer captures the current value of the count register (= TMxCNT ÷ 2) and transfers it into the TMxW width registers. At the next leading edge, the timer transfers the current value of the count register (= TMxCNT ÷ 2) into the TMxPRD period register. The count registers are reset to 0x0000 0000 again, and the timer continues counting until it is either disabled or the count value reaches 0xFFFF FFFF.

In this mode, programs can measure both the pulse width and the pulse period of a waveform. To control the definition of the leading edge and

trailing edge of the TIMERx_I signal, the PULSE bit in the TMxCTL register is set or cleared. If the PULSE bit is cleared, the measurement is initiated by a falling edge, the count register is captured to the WIDTH register on the rising edge, and the period register is captured on the next falling edge.

The PRDCNT bit in the TMxCTL register controls whether an enabled interrupt is generated when the pulse width or pulse period is captured. If the PRDCNT bit is set, the interrupt latch bit (TIMxIRQ) gets set when the pulse period value is captured. If the PRDCNT bit is cleared, the TIMxIRQ bit gets set when the pulse width value is captured.

If the PRDCNT bit is cleared, the first period value has not yet been measured when the first interrupt is generated. Therefore, the period value is not valid. If the interrupt service routine reads the period value anyway, the timer returns a period value of zero. When the period expires, the period value is loaded in the TMxPRD register.

A timer interrupt (if enabled) is also generated if the count register reaches a value of 0xFFFF FFFF. At that point, the timer is disabled automatically, and the TIMxOVF status bit is set, indicating a count overflow. The TIMxIRQ and TIMxOVF bits are sticky bits, and programs must explicitly clear them. The WDTH_CAP timing is shown in Figure 9-6.

The first width value captured in WDTH_CAP mode is erroneous due to synchronizer latency. To avoid this error, programs must issue two NOP instructions between setting WDTH_CAP mode and setting TIMxEN.

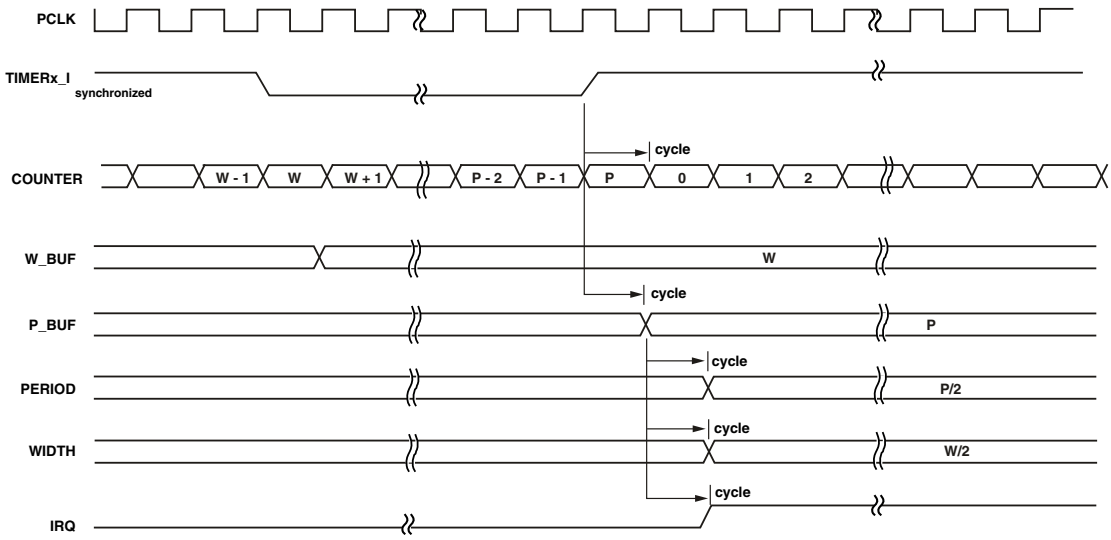


Figure 9-6. WDTH_CAP Timing (Period Count = 1)

External Event Watchdog Mode (EXT_CLK)

Figure 9-7 shows a flow diagram for EXT_CLK mode. To enable EXT_CLK mode, set the TIMODE1-0 bits in the TMxCTL register to 11 in the TMxCTL register. This samples the TIMERx_I signal as an input. Therefore, in EXT_CLK mode, the TMxCNT register should not be read when the counter is running.

The operation of the EXT_CLK mode is as follows:

1. Program the TMxPRD period register with the value of the maximum timer external count.
2. Set the TIMxEN bits. This loads the period value in the count register and starts the countdown.
3. When the period expires, an interrupt, (TIMxIRQ) occurs.

After the timer is enabled, it waits for the first rising edge on the TIMERx_I signal. The rising edge forces the count register to be loaded by the value (0xFFFF FFFF – TMxPRD). Every subsequent rising edge increments the count register. After reaching the count value 0xFFFF FFFE, the TIMxIRQ bit is set and an interrupt is generated. The next rising edge reloads the count register with (0xFFFF FFFF – TMxPRD) again.

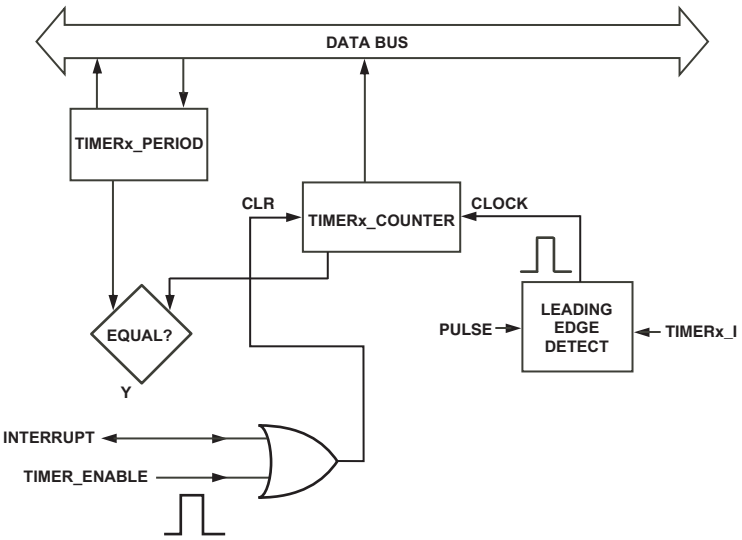


Figure 9-7. Flow Diagram EXT_CLK Mode

For EXT_CLK mode only, setting the PULSE bit to 1 or 0 does not have any effect on the edge in which the count happens. It is always clocked at the rising edge.

The EXT_CLK timing is shown in Figure 9-8.

The configuration bit, PRDCNT, has no effect in this mode. Also, TIMxOVF is never set and the width register is unused.

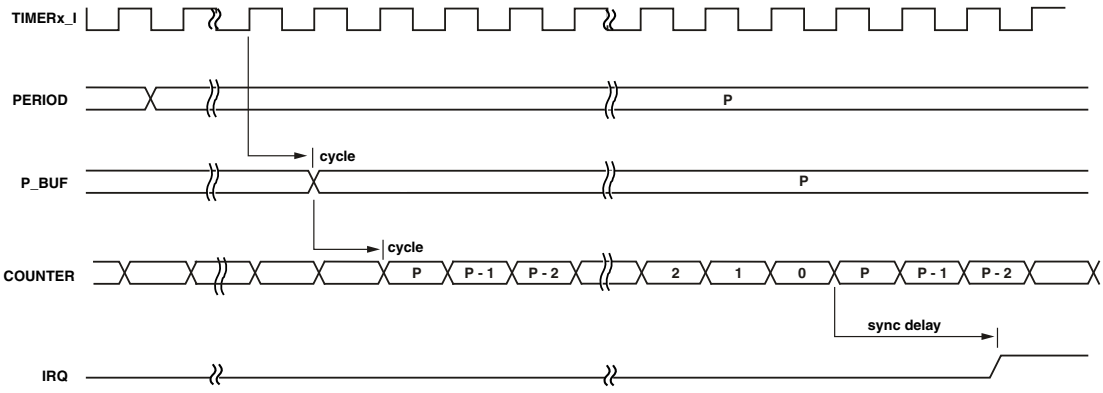


Figure 9-8. EXT_CLK Timing

Interrupts

This section describes all relevant registers and hardware to raise and service interrupts.

Sources

Each timer generates a unique interrupt request signal. A common register latches these interrupts so that a program can determine the interrupt source without reference to the timer's interrupt signal. The `TMSTAT` register contains an interrupt latch bit (`TIMxIRQ`) and an overflow/error indicator bit (`TIMxOVF`) for each timer.

These sticky bits are set by the timer hardware and may be watched by software. They need to be cleared in the `TMSTAT` register by software explicitly. To clear, write a one to the corresponding bit in the `TMSTAT` register as in the following example.

```
TMR0_ISR:
      bit set ustat2 TIM0IRQ;   /* W1C the Timer0 bit */
      dm(TM0STAT)=ustat2;
      instructions;
      instructions;
      instructions;
      RTI;
TMR0_ISR.end:
```

Interrupt and overflow bits may be cleared simultaneously with timer enable or disable.

To enable a timer's interrupt, set the IRQEN bit in the timer's configuration (TMxCTL) register and unmask the timer's interrupt by setting the corresponding bit of the IMASK register. With the IRQEN bit cleared, the timer does not set its interrupt latch (TIMxIRQ) bits. To poll the TIMxIRQ bits without generating a timer interrupt, programs can set the IRQEN bit while leaving the timer's interrupt masked.

With interrupts enabled, ensure that the interrupt service routine (ISR) clears the TIMxIRQ latch before the RTI instruction to assure that the interrupt is not serviced erroneously. In external clock (EXT_CLK) mode, the latch should be reset at the very beginning of the interrupt routine so as not to miss any timer event.

Watchdog Functionality

Any of the timers can be used to implement a watchdog functionality that can be controlled by either an internal or an external clock source.

For a program to service the watchdog, the program must reset the timer value by disabling and then re-enabling the timer. Servicing the watchdog periodically prevents the count register from reaching the period valueand prevents the timer interrupt from being generated. When the timer reaches the period value and generates the interrupt, reset the processor within the corresponding watchdog's ISR.

Effect Latency

The timer starts 3 PCLK cycles after the TIMEN bit is set.

When the timer is enabled, the count register is loaded according to the operation mode specified in the TMxCTL register. When the timer is disabled, the counter registers retain their state; when the timer is re-enabled, the counter is reinitialized based on the operating mode. The program should never write the counter value directly.

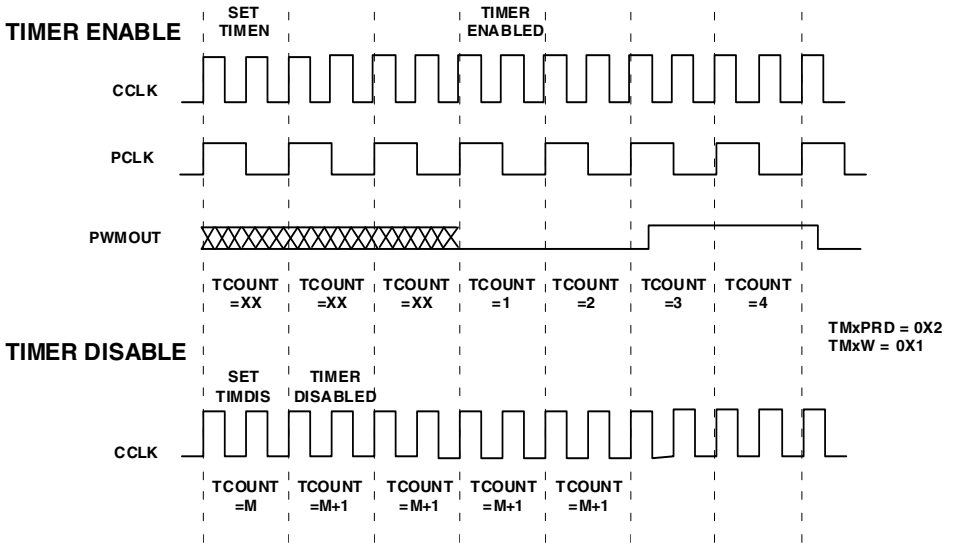


Figure 9-9. Timer PWM Enable and Disable Timing

Debug Features

The following section provides information on debugging features available with the timer. Note that in emulation space during a core halt that the timer continues to operate.

Loopback Routing

The timer support an internal loopback mode by using the SRU. For more information, see “Loop Back Routing” on page 5-30.

Programming Model

The section describes which sequences ofsoftware steps are required to get the peripheral working successfully.

PWM Out Mode

Use the following procedure to configure and run the timer in PWM out mode.

1. Reset the TIMEN bit and set the configuration mode to 01 to select PWM_OUT operation. This configures the TIMERx_O pin as an output pin with its polarity determined by the PULSE bit.
    - Measures a positive active pulse width at the TIMERx_O pin.
    - Measures a negative active pulse width at the TIMERx_O pin.
2. Initialize the period and width register values. Insure that the period value is greater than the width value.
3. Set the TIMEN bit. The timer performs boundary exception checks on the period and width values:
    - If (width == 0 or Period < width or period == width) both the OVF_ERR and $\overline{\text{IRQ}}$ bits are set.
    - If there are no exceptions, thewidth value is loaded into the counter and it starts counting.

The timer produces PWM waveform with a period of 2 x period and a width of 2 x width.

- When 2 x width expires, the counter is loaded with 2x(period – width) and continues counting.
- When 2 x period expires, the counter is loaded with 2 x width value again and the cycle repeats.
- When the width or period expires, the $\overline{\text{IRQ}}$ bit (if enabled) is set depending on the PRDCNT bit.
- When $\overline{\text{IRQ}}$ is sensed, read the status register (TMxSTAT) and perform the appropriate “write-one” to clear.

WDTH_CAP Mode

Use the following procedure to configure and run the timer in WDTH_CAP out mode.

1. Reset the TIMEN bit and set the configuration mode to 10 to select WDTH_CAP operation. This configures the TIMERx_I pin as an input pin with its polarity determined by the PULSE bit.
   - Measures a positive active pulse width at the TIMERx_I pin.
   - Measures a negative active pulse width at the TIMERx_I pin.
2. The PRDCNT bit determines when the $\overline{\text{IRQ}}$ status bit (if enabled) is set.
   - If PRDCNT == 1), $\overline{\text{IRQ}}$ is set when the period expires and the value is captured.
   - If PRDCNT == 0), $\overline{\text{IRQ}}$ is set when the width expires and the value is captured.

3. Valid period and width values are set in their respective registers when IRQ is set.

   The period and width values are measured with respect to PCLK. This makes this mode coherent with the PWM_OUT mode, where the output waveforms have a period of 2 x period and a width of 2 x width.

   Note that the first period value will not have been measured when the first width is measured, so it is not valid. The timer sets and returns a period value of zero in this case. When the period expires, the period value is placed into the period register. When IRQ is sensed, read the status and perform the appropriate "write-one" to clear.

EXT_CLK Mode

Use the following procedure to configure and run the timer in EXT_CLK out mode.

1. Reset the TIMEN bit and set the configuration mode to 11 to select EXT_CLK operation.

   This configures the TIMERx_I pin as an input pin regardless of the setting of the PULSE bit. Note that the timer always samples the rising edge in this mode. The period register is WO and the width register is unused in this mode.

2. Initialize the period register with the value of the maximum external count.

3. Set the TIMEN bit. This loads the period value in the counter and starts the count down.

   When the period expires, it is reloaded with the period value and the cycle repeats. Counter counts with each edge of the input waveform, asynchronous to PCLK.

When the period expires, $\overline{\text{IRQ}}$ (if enabled) is set and TMR_IRQ is asserted. An external clock can trigger the Timer to issue an interrupt and wake up an idle processor.

Reads of the count register are not supported in EXT_CLK mode.

Programming Examples

This section provides two programming examples written for the ADSP-2136x processor processors.

The first listing, Listing 9-1, sets up timer 0 in external watchdog mode, using DAI pin 1 as its input. The timer generates an interrupt when it senses the number of edges are equal to the timer period setting. The second listing, Listing 9-2, uses both timer 0 and timer 1. Timer 0 is set up in PWMOUT mode, using DAI pin 1 as its output. Timer 1 is set up in width capture mode, using Timer 0 as its input. The period and pulse width measured by timer 1 are identical to the settings of timer 0.

Listing 9-1. External Watchdog Mode Example

```
#include <def21364.h>
#include <sru21364.h>
#include <SRU.h>

/* Route Timer0 Input to DAI Pin 1 via SRU */

SRU(DAI_PB01_O, TIMER0_I);

ustat3 = TIMODEEXT|        /* External Watchdog Mode */
                           /* Positive edge always active */
IRQEN|                     /* Enable Timer 0 interrupt */
PRDCNT;                    /* Count to end of period */
dm(TMOCTL) = ustat3;
```

```
R0 = 0xff;
dm(TM0PRD) = R0;           /* Timer 0 period = 255 */

/* An interrupt is generated when the Timer senses end of the
selected period */
R0 = TIM0EN;                    /* Enable timer 0 */
dm(TMSTAT) = R0;
_main.end: jump (pc,0);          /* endless loop */
```

Listing 9-2. PWMOUT and Width Capture Mode Example

```
/* Set up and enable Timer 0 in PWM Out mode*/
/* Route Timer0 Output to DAI Pin 1 via SRU */

SRU(TIMER0_O, DAI_PB01_I);

/* Enable DAI pin 1 as an output */
SRU(HIGH, PBEN01_I);

ustat3 = TIM0DEPWM|         /* PWM Out Mode */
PULSE|                      /* Positive edge is active */
PRDCNT;                     /* Count to end of period */
dm(TM0CTL) = ustat3;
R0 = 0xFF;
dm(TM0PRD) = R0;            /* Timer 0 period = 255 */
R1 = 0x3F;
dm(TM0W) = R1;              /* Timer 0 Pulse width = 15 */
R0 = TIM0EN;                /* enable timer 0 */
dm(TMSTAT) = R0;

/* --------------End of Timer 0 Setup-------------------- */
/* Set up and enable Timer 1 in Width Capture mode */
/* Use the output of Timer 0 as the input to Timer 1 */
/* Route Timer 0 Output to Timer 1 Input via SRU */
```

```
SRU(TIMER0_O, TIMER1_I);

ustat3 = TIMODEW|     /* Width Capture mode */
PULSE|                /* Positive edge is active */
IRQEN|                /* Enable Timer 1 Interrupt */
PRDCNT;               /* Count to end of period */
dm(TM1CTL) = ustat3;
R0 = TIM1EN;          /* enable timer 1 */
dm(TMSTAT) = R0;

/* Poll the Timer 1 interrupt latch, the interrupt will latch
when the measured period and pulse width are ready to read */

bit tst LIRPTL GPTMR1I;
if not tf jump(pc,-1);

/* Read the measured values */
r0 = dm(TM1PRD);
r1 = dm(TM1W);

/* r0 and r1 will match the Timer 0 settings above */
_main.end: jump (pc,0);
```

10 PULSE WIDTH MODULATION

Pulse width modulation (PWM) is a technique for controlling analog circuits with a microprocessor's digital outputs. PWM is employed in a wide variety of applications, ranging from measurement to communications to power control and conversion.

Features

Table 10-1 provides a brief summary of the features of this interface.

Table 10-1. PWM Feature Summary

Feature	Availability
Connectivity	
Multiplexed Pinout	Yes, (Parallel Port)
SRU DAI Required	No
SRU DAI Default Routing	N/A
Interrupt Default Routing	Yes (P13I)
Protocol	
Master Capable	Yes
Slave Capable	N/A
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A

Table 10-1. PWM Feature Summary (Cont'd)

Feature	Availability
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Interrupt Source	Core
Boot Capable	N/A
Local Memory	No
Clock Operation	PCLK

- Four independent PWM units
- Center aligned PWM
- Edge aligned PWM
- 2-phase output timing unit

One of the advantages of PWM is that the signal remains digital all the way from the processor to the controlled system; no digital-to-analog conversion is necessary. By maintaining a digital signal throughout a system, noise effects are minimized.

The PWM module in the ADSP-2136x processor is a flexible, programmable, PWM waveform generator. It is capable of generating switching patterns for various purposes such as motor control, electronic valve control, or audio power control. The module can generate either center-aligned or edge-aligned PWM waveforms. In addition, it can generate complementary signals on two outputs in paired mode or inde-

pendent signals in non-paired mode. A block diagram of the module appears in Figure 10-1.

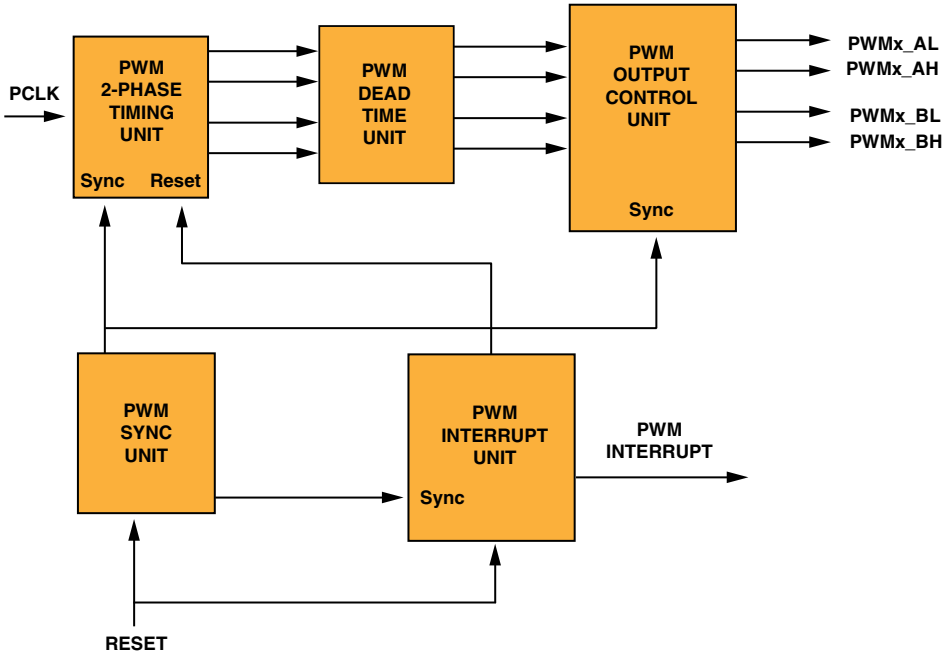


Figure 10-1. PWM Module Block Diagram

Pin Descriptions

The PWM module has four groups of four PWM outputs each, for a total of 16 PWM outputs. These outputs are described in Table 10-2.

Table 10-2. PWM Pin Descriptions

Multiplexed Pin Name	Direction	Description
PWM_AH3–0	O	PWM output of pair A produce high side drive signals.
PWM_AL3–0	O	Complementary PWM output of pair A produce low side drive signals.
PWM_BH3–0	O	PWM output of pair B produce high side drive signals.
PWM_BL3–0	O	Complementary PWM output of pair B produce low side drive signals.

Functional Description

Each PWM group is able to generate complementary signals on two outputs in paired mode or each group can provide independent outputs in non-paired mode.

The switching frequency and dead time (see "Dead Time" on page 10-12) of the generated PWM patterns are programmable using the PWMPERIODx and PWMDTx registers. In addition, two duty cycle control registers (PWMAx and PWMBx) directly control the duty cycles of the two pairs of PWM signals. In non-paired mode, the low side signals can have different duty cycles programmed through another pair of registers (PWMALx and PWMBLx). It should be further noted that the choice of center- or edge-aligned mode applies to a single group of four PWM waveforms. Each of the four PWM output signals can be enabled or disabled by separate output enable bits in the PWMSEG0-3 register (see "PWM Output Disable Registers (PWMSEGx)" on page A-27). Additionally, in center-aligned paired mode, an emergency dead time insertion circuit enforces a dead time defined by

PWMDT0-3 between the high and low side drive signals of each PWM channel. This ensures the correct dead time occurs at the power inverter. In many applications, there is a need to provide an isolation barrier in the gate drive circuits that turn on the power devices of the inverter.

The fundamental timing unit of the PWM controller is peripheral clock (PCLK).

Register Descriptions

The registers described below control the operation and provide the status of pulse width modulation on the ADSP-2136x processor. For more information, see “Pulse Width Modulation Registers” on page A-23.

- **PWM global control register.** The PWMGCTL register enables or disables the four PWM groups in any combination.
- **PWM control registers.** The PWMCTL3-0 registers are used to set the operating modes of each PWM block. This register also allows programs to disable interrupts from individual groups.
- **PWM period registers.** The PWMPERIOD3-0 registers are 16-bit read-write registers that control the period of the four PWM groups.
- **PWM dead time registers.** The PWMDT3-0 registers are 16-bit read-write registers that are used to set the switching dead time.
- **PWM channel A and B duty control registers.** The PWMA3-0 and PWMB3-0 registers directly control the duty cycles of the two pairs of PWM output signals on the PWM_AxH to PWM_Bx pins when not in switch reluctance mode.
- **PWM output enable registers.** The PWMSEG3-0 registers are 16-bit read-write registers that are used to control the output signals of the four PWM groups.

- **PWM channel A and B low side duty control registers.** In non-paired mode, the PWMAL3-0 and PWMBL3-0 registers are used to program the low side duty cycle of the two-pairs of PWM output signals. These can be different on the high side cycles.
- **PWM output polarity select registers.** The PWMPOL3-0 registers are 16-bit read/write registers that are used to determine whether the polarity of the generated PWM signals is active high or active low. The polarity values can be changed on the fly if required, provided the change is done a few cycles before the next period change.
- **PWM global status register.** The PWMGSTAT register provides the status of each PWM group. The bits in this register are W1C type (write one to clear).
- **PWM status registers.** The PWMSTAT3-0 registers are 16-bit read-only registers that report the phase and mode status for each PWM group.
- **PWM debug status registers.** The PWMDBG3-0 registers are 16-bit read-only registers that report the output pin status for each PWM group.

Operation Modes

The following sections provide information on the operating modes of the PWM module.

Groups Synchronization

The PWMGCTL register enables or disables the four PWM groups in any combination. This provides synchronization across the four PWM groups.

The PWM_SYNC_ENx bits in this register can be used to start the counter without enabling the outputs through PWM_EN. So when PWM_ENx is

asserted, the 4 PWM outputs are automatically synced to the initially programmed period. In most cases, all SYNC bits can be initialized to zero, enabling the PWM_ENx bits of the four PWM groups at the same time synchronizes the four groups.

The PWM sync enable feature allows programs to enable the PWN_SYNC_ENx bits to independently start the main counter without enabling the corresponding PWM module using the PWM_ENx bits. To synchronize different groups, enable the corresponding group's PWM_ENx bit at the same time. In order to stop the counter both the PWM_DISx and PWM_SYNC_DISx bits should be set in this register.

PWM Timer

The internal operation of the PWM generation unit is controlled by the PWM timer which is clocked at the peripheral clock rate, PCLK. The operation of the PWM timer over one full PWM period is illustrated in Figure 10-2. It can be seen that during the first half cycle (PWMSTAT bit PWMPHASE is cleared), the PWM timer decrements from PWMPERIOD/2 to –PWMPERIOD/2 using a two's complement count.

At this point, the count direction changes and the timer continues to increment from –PWMPERIOD/2 to the PWMPERIOD/2 value. Of course, the value of the PWMPERIOD register could be altered at the mid-point in double update mode. In such a case, the duration of the second half period (PWMSTAT bit PWMPHASE is set) may be different then that of the first half cycle. The PWMPERIOD is double buffered and a change in one half of the PWM switching period only takes effect in the next half period.

The PWM module on the SHARC processor can generate waveforms that are either edge-aligned (left-justified) or center-aligned. Each waveform is described in detail in the following sections.

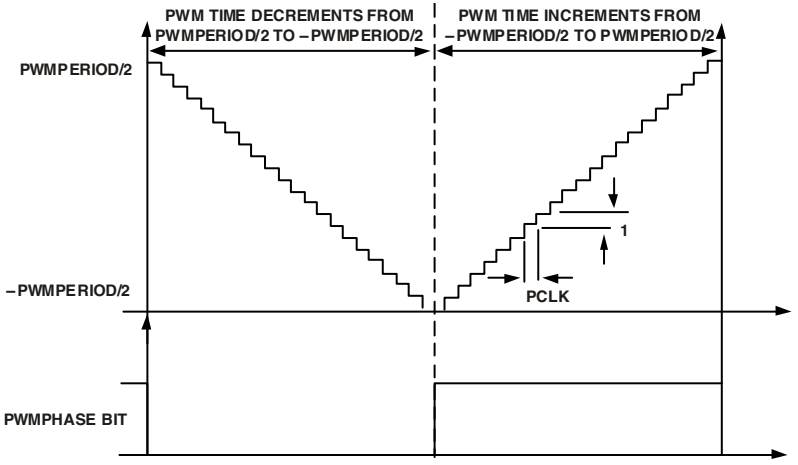


Figure 10-2. Operation of Internal PWM Timer

Edge-Aligned Mode

In edge-aligned mode, shown in Figure 10-3, the PWM waveform is left-justified in the period window. A duty value of zero, programmed through the PWMAx registers, produces a PWM waveform with 50% duty cycle. For even values of period, the PWM pulse width is exactly period/2, whereas for odd values of period, it is equal to period/2 (rounded up). Therefore for a duty value programmed in two's-compliment, the PWM pulse width is given by:

$$Width = \lceil (period) \div 2 \rceil + duty$$

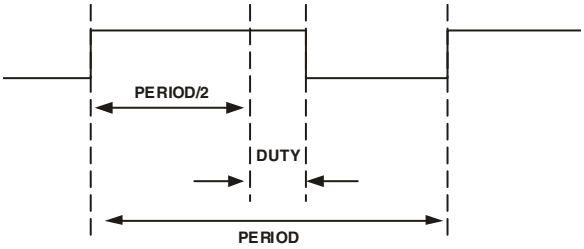


Figure 10-3. Edge Aligned PWM Wave with High Polarity

To generate constant logic high on PWM output, program the duty register with the value ≥ + period/2.

To generate constant logic low on PWM output, program the duty register with the value ≥ – period/2.

For example, using an odd period of p = 2n + 1, the counter within the PWM generator counts as (–n...0...+n). If the period is even (p = 2n) then the counter counts as (–n+1...0...n).

For edge aligned mode: $f_{PWM} = f_{PCLK}$/PWMPERIODx.

For more information, see “PWM Channel Duty Control Registers (PWMAx, PWMBx)” on page A-29.

Center-Aligned Mode

Most of the following description applies to paired mode, but can also be applied to non-paired mode, the difference being that each of the four outputs from a PWM group is independent. Within center aligned mode, there are several options to choose from.

Center-Aligned Single-Update Mode. Duty cycle values are programmable only once per PWM period, so that the resultant PWM patterns are symmetrical about the mid-point of the PWM period.

Center-Aligned Double-Update Mode. Duty cycle values are programmable only twice per PWM period. This second updating of the PWM registers is implemented at the mid-point of the PWM period, producing asymmetrical PWM patterns that produce lower harmonic distortion in two-phase PWM inverters.

Center-Aligned Paired Mode. Generates complementary signals on two outputs.

Center-Aligned Non-Paired Mode. Generates independent signals on two outputs.

In paired mode, the two's-complement integer values in the 16-bit read/write duty cycle registers, PWMAx and PWMBx, control the duty cycles of the four PWM output signals on the PWM_AL, PWM_AH, PWM_BL and PWM_BH pins respectively. The duty cycle registers are programmed in two's-complement integer counts of the fundamental time unit, PCLK and define the desired on time of the high side PWM signal over one-half the PWM period.

The duty cycle register range is from (–PWMPERIOD/2 – PWMDT) to (+PWMPERIOD/2 + PWMDT), which, by definition, is scaled such that a value of 0 represents a 50% PWM duty cycle.

Each group in the PWM module (0–3) has its own set of registers which control the operation of that group. The operating mode of the PWM block (single or double update mode) is selected by the PWM_UPDATE bit (bit 2) in the PWM control (PWMCTRL3-0) registers. Status information about each individual PWM group is available to the program in the PWM status (PWMSTAT3-0) registers. Apart from the local control and status registers for each PWM group, there is a single PWM global control register (PWMGCTL) and a single PWM global status register (PWMGSTAT). The global control register allows programs to enable or disable the four groups in any combination, which provides synchronization across the four PWM groups.

The global status register shows the period completion status of each group. On period completion, the corresponding bit in the PWMGSTAT register is set and remains sticky. The program first reads the global status register and clears all the intended bits by explicitly writing 1.

Switching Frequencies

The 16-bit read/write PWM period registers, PWMPERIOD3-0, control the PWM switching frequency.

The fundamental timing unit of the PWM controller is PCLK. Therefore, for a 100 MHz peripheral clock, the fundamental time increment is 10 ns. The value written to the PWMPERIODx register is effectively the number of PCLK clock increments in half a PWM period. The required PWMPERIODx value as a function of the desired PWM switching frequency (f_{PWM}) is given by:

$$PWMPERIOD = \frac{f_{PCLK}}{2 \times f_{PWM}}$$

Therefore, the PWM switching period, T_s, can be written as:

$$T_s = 2 \times PWMTM \times t_{PCLK}$$

For example, for a 100 MHz PCLK and a desired PWM switching frequency of 10 kHz (T_s = 100 µs), the correct value to load into the PWMPERIODx register is:

$$PWMPERIOD = \frac{100 \times 10^6}{2 \times 10 \times 10^3} = 5000$$

The largest value that can be written to the 16-bit PWMPERIODx register is 0xFFFF = 65,535 which corresponds to a minimum PWM switching frequency of:

$$f_{(PWM),min} = \frac{100 \times 10^6}{2 \times 65535} = 763Hz$$

PWMPERIOD values of 0 and 1 are not defined and should not be used when the PWM outputs or PWM sync is enabled.

Dead Time

The second important parameter that must be set up in the initial configuration of the PWM block is the switching dead time. This is a short delay time introduced between turning off one PWM signal (say AH) and turning on the complementary signal, AL. This short time delay is introduced to permit the power switch being turned off (AH in this case) to completely recover its blocking capability before the complementary switch is turned on. This time delay prevents a potentially destructive short-circuit condition from developing across the DC link capacitor of a typical voltage source inverter.

The 10-bit, read/write PWMDT3-0 registers control the dead time. The dead time, T_d, is related to the value in the PWMDTx registers by:

$$T_d = PWMDT \times 2 \times t_{PCLK}$$

Therefore, a PWMDT value of 0x00A (= 10), introduces a 200 ns delay between when the PWM signal (for example AH) is turned off and its complementary signal (AL) is turned on. The amount of the dead time can therefore be programmed in increments of 2 × PCLK (or 20 ns for a 100 MHz peripheral clock). The PWMDTx registers are 10-bit registers, and the maximum value they can contain is 0x3FF (= 1023) which corresponds to

a maximum programmed dead time of:

$$Td,\ max\ =\ 1023 \times 2 \times t_{PCLK}\ =\ 1023 \times 2 \times 10 \times 10^{-9} =\ 20.5\ micro\ sec$$

This equates to an PCLK rate of 100 MHz. Note that dead time can be programmed to zero by writing 0 to the PWMDTx registers (see "PWM Dead Time Registers (PWMDTx)" on page A-29).

Duty Cycles

The two 16-bit read/write duty cycle registers, PWMA and PWMB, control the duty cycles of the four PWM output signals on the PWM pins when not in switch reluctance mode. The two's-complement integer value in the PWMA register controls the duty cycle of the signals on the PWM_AH and PWM_AL. The two's-complement integer value in the PWMB register controls the duty cycle of the signals on PWM_BH and PWM_BL pins. The duty cycle registers are programmed in two's-complement integer counts of the fundamental time unit, PCLK, and define the desired on-time of the high-side PWM signal produced by the two-phase timing unit over half the PWM period. The duty cycle register range is from:

(–*PWPERIOD* ÷ 2 – *PWMDT*) to (+*PWPERIOD* ÷ 2 + *PWMDT*)

which, by definition, is scaled such that a value of 0 represents a 50% PWM duty, cycle. The switching signals produced by the two-phase timing unit are also adjusted to incorporate the programmed dead time value in the PWMDT register. The two-phase timing unit produces active low signals so that a low level corresponds to a command to turn on the associated power device.

Duty Cycles and Dead Time

A typical pair of PWM outputs (in this case for PWM_AH and PWM_AL) from the timing unit are shown in Figure 10-4 for operation in single-update

mode. All illustrated time values indicate the integer value in the associated register and can be converted to time by simply multiplying by the fundamental time increment, (PCLK) and comparing this to the two's-complement counter. Note that the switching patterns are perfectly symmetrical about the midpoint of the switching period in single-update mode since the same values of the PWMAx, PWMPERIODx, and PWMDTx registers are used to define the signals in both half cycles of the period.

Further, the programmed duty cycles are adjusted to incorporate the desired dead time into the resulting pair of PWM signals. As shown in Figure 10-4, the dead time is incorporated by moving the switching instants of both PWM signals (PWM_AH and PWM_AL) away from the instant set by the PWMAx registers. Both switching edges are moved by an equal amount (PWMDT x PCLK) to preserve the symmetrical output patterns. Also shown is the PWM_PHASE bit of the PWMSTAT register that indicates whether operation is in the first or second half cycle of the PWM period.

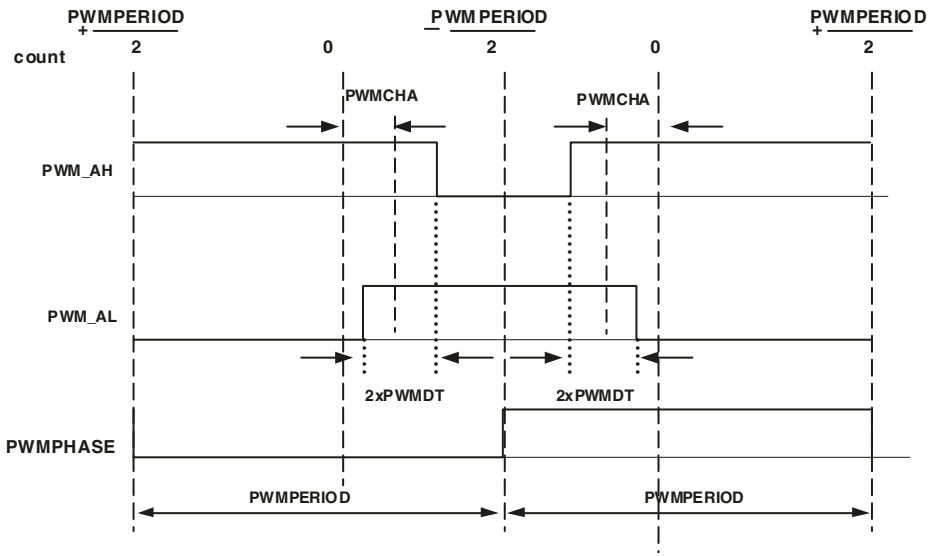


Figure 10-4. Center-Aligned Paired PWM in Single-Update Mode, Low Polarity

The resulting on-times (active low) of the PWM signals over the full PWM period (two half periods) produced by the PWM timing unit and illustrated in Figure 10-5 on page 10-16 may be written as:

The range of T_{AH} is:

$$[0 - 2 \times PWMPERIOD \times t_{PCLK}]$$

and the corresponding duty cycles are:

$$T_{AH} = (PWMPERIOD + 2 \times (PWMCHA - PWMDT)) \times t_{PCLK}$$

The range of T_{AL} is:

$$T_{AL} = (PWMPERIOD - 2 \times (PWMCHA + PWMDT)) \times t_{PCLK}$$

and the corresponding duty cycles are:

$$d_{AH} = \frac{T_{AH}}{T_S} = \frac{1}{2} + \frac{PWMCHA - PWMDT}{PWMPERIOD}$$

$$d_{AL} = \frac{T_{AL}}{T_S} = \frac{1}{2} - \frac{PWMCHA + PWMDT}{PWMPERIOD}$$

The minimum permissible value of T_{AH} and T_{AL} is zero, which corresponds to a 0% duty cycle, and the maximum value is T_S, the PWM switching period, which corresponds to a 100% duty cycle. Negative values are not permitted.

The output signals from the timing unit for operation in double-update mode are shown in Figure 10-5. This illustrates a general case where the switching frequency, dead time, and duty cycle are all changed in the sec-

ond half of the PWM period. The same value for any or all of these quantities can be used in both halves of the PWM cycle. However, there is no guarantee that a symmetrical PWM signal will be produced by the timing unit in this double-update mode Additionally, Figure 10-5 shows that the dead time is inserted into the PWM signals in the same way as in single-update mode.

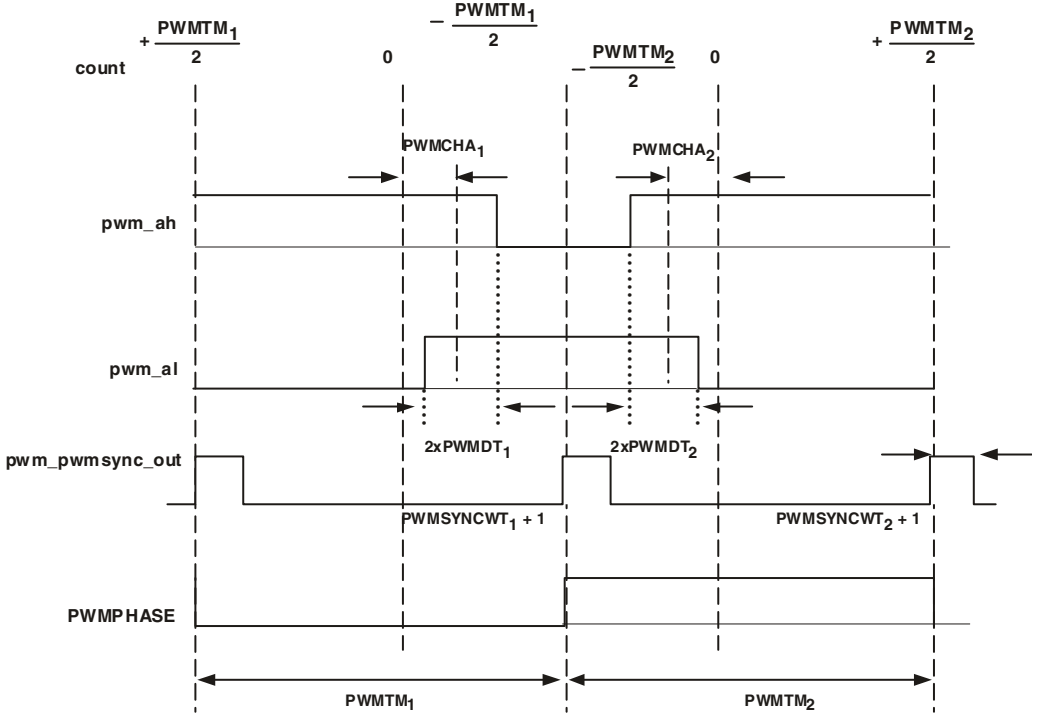


Figure 10-5. Center-Aligned Paired PWM in Double-Update Mode, Low Polarity

In general, the on-times (active low) of the PWM signals over the full PWM period in double-update mode can be defined as:

$$T_S = (PWMPERIOD_1 + PWMPERIOD_2) \times t_{PCLK}$$

$$T_{AH} = \left(\frac{PWMPERIOD_1}{2} + \frac{PWMPERIOD_2}{2} + PWMCHA_1 + PWMCHA_2 - PWMDT_1 - PWMDT_2\right) \times t_{PCLK}$$

$$T_{AL} = \left(\frac{PWMPERIOD_1}{2} + \frac{PWMPERIOD_2}{2} - PWMCHA_1 - PWMCHA_2 - PWMDT_1 - PWMDT_2\right) \times t_{PCLK}$$

where subscript 1 refers to the value of that register during the first half cycle and subscript 2 refers to the value during the second half cycle. The corresponding duty cycles are:

$$d_{AH} = \frac{T_{AH}}{T_S} = \frac{1}{2} + \frac{(PWMCHA_1 + PWMCHA_2 - PWMDT_1 - PWMDT_2)}{(PWMPERIOD_1 + PWMPERIOD_2)}$$

$$d_{AL} = \frac{T_{AL}}{T_S} = \frac{1}{2} - \frac{(PWMCHA_1 + PWMCHA_2 + PWMDT_1 + PWMDT_2)}{(PWMPERIOD_1 + PWMPERIOD_2)}$$

since for the general case in double- update mode, the switching period is given by:

$$T_S = (PWMPERIOD_1 + PWMPERIOD_2) \times t_{PCLK}$$

Again, the values of T_{AH} and T_{AL} are constrained to lie between zero and T_S. Similar PWM signals to those illustrated in Figure 10-4 and Figure 10-5 can be produced on the BH and BL outputs by programming the `PWMBx` registers in a manner identical to that described for the `PWMAx` registers.

Over-Modulation

The PWM timing unit is capable of producing PWM signals with variable duty cycle values at the PWM output pins. At the extreme side of the modulation process, settings of 0% and 100% modulation are possible. These two modes are termed full OFF and full ON respectively. Settings that fall between the extremes are considered normal modulation. These settings are explained in more detail below.

Full On. The PWM for any pair of PWM signals operates in full on when the desired high side output of the two-phase timing unit is in the on state (low) between successive PWMSYNC rising edges. This state may be entered by virtue of the commanded duty cycle values in conjunction with the setting in the PWMDTx registers.

Full Off. The PWM for any pair of PWM signals operates in full off when the desired high side output of the two-phase timing unit is in the off state (high) between successive PWMSYNC pulses. This state may be entered by virtue of the commanded duty cycle values in conjunction with the setting in the PWMDTx registers.

Normal Modulation. The PWM for any pair of PWM signals operates in normal modulation when the desired output duty cycle is other than 0% or 100% between successive PWMSYNC pulses.

There are certain situations, when transitioning either into or out of either full on or full off, where it is necessary to insert additional *emergency dead time* delays to prevent potential *shoot-through conditions* in the inverter. These transitions are detected automatically and, if appropriate, the emergency dead time is inserted to prevent the shoot through conditions.

Inserting additional emergency dead time into one of the PWM signals of a given pair during these transitions is only needed if both PWM signals would otherwise be required to toggle within a dead time of each other. The additional emergency dead time delay is inserted into the PWM signal that is toggling into the on state. In effect, the turn on (if turning on during this dead time region), of this signal is delayed by an amount of

2 x PWMDT x PCLK from the rising edge of the opposite output. After this delay, the PWM signal is allowed to turn on, provided the desired output is still scheduled to be in the on state after the emergency dead time delay.

Figure 10-6 illustrates two examples of such transitions. In (a), when transitioning from normal modulation to full on at the half cycle boundary in double-update mode, no special action is needed. However in (b), when transitioning into full off at the same boundary, an additional emergency dead time is necessary. This inserted dead time is a little different to the normal dead time as it is impossible to move one of the switching events back in time because this would move the event into the previous modulation cycle. Therefore, the entire emergency dead time is inserted by delaying the turn on of the appropriate signal by the full amount.

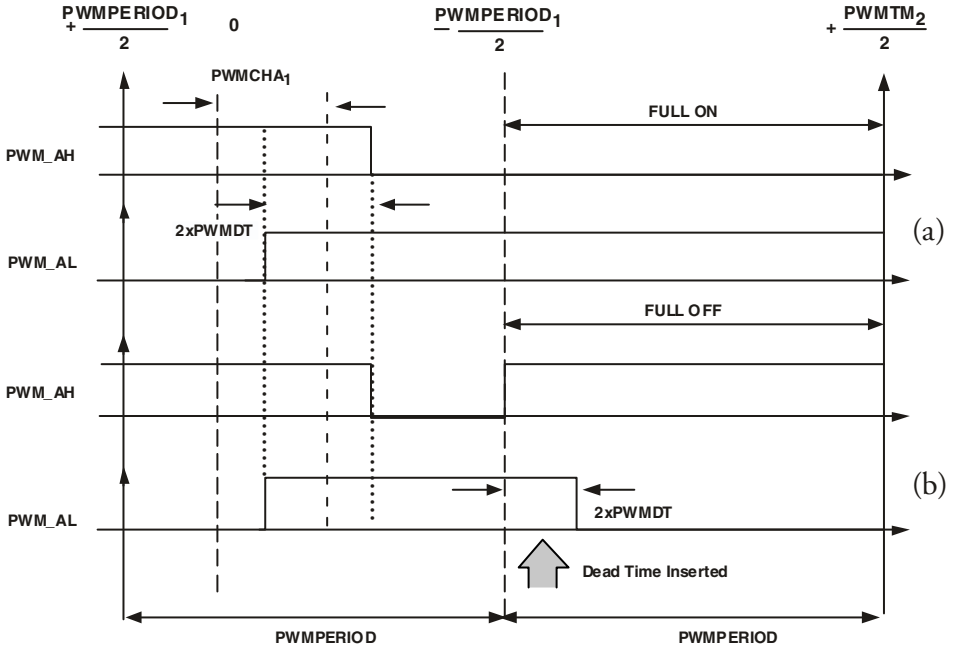


Figure 10-6. Full On to Full Off Transition

Update Modes

Update modes determine the frequency with which the wave forms are sampled.

Single-Update

In this mode, duty cycle values are programmable only once per PWM period, so that the resultant PWM patterns are symmetrical at the mid-point of the PWM period.

Double-Update

In this mode, a second updating of the PWM registers is implemented at the mid-point of the PWM period. In this mode, it is possible to produce asymmetrical PWM patterns that produce lower harmonic distortion in two-phase PWM inverters. This technique also permits closed loop controllers to change the average voltage applied to the machine windings at a faster rate and so permits faster closed loop bandwidths.

Configuring Polarity

The polarity of the generated PWM signals is programmed using the PWMPOLARITY3-0 registers (see "PWM Polarity Select Registers (PWMPOLx)" on page A-28), so that either active high or active low PWM patterns can be produced. The polarity values can be changed on the fly if required, provided the change is done a few cycles before the next period change.

Accuracy

The PWM has 16-bit resolution but accuracy is dependent on the PWM period. In single-update mode, the same values of PWMA and PWMB are used to define the on times in both half cycles of the PWM period. As a result, the effective accuracy of the PWM generation process is 2 x PCLK (or 20 ns for a 100 MHz clock). Incrementing one of the duty cycle registers by one changes the resultant on time of the associated PWM signals by 2 x PCLK in each half period (or 2 x PCLK for the full period). In double-update mode, improved accuracy is possible since different values of the duty cycles registers are used to define the on times in both the first and second halves of the PWM period. As a result, it is possible to adjust the on-time over the whole period in increments of PCLK. This corresponds to an effective PWM accuracy of PCLK in double-update mode (or 10 ns for a 100 MHz clock). The achievable PWM switching frequency at a given PWM accuracy is tabulated in Table 10-3.

Table 10-3. PWM Accuracy in Single- and Double-Update Modes

Resolution (bits)	Single-Update Mode PWM Frequency (kHz)	Double-Update Mode PWM Frequency (kHz)
8	195.3	390.6
9	97.7	195.3
10	48.8	97.7
11	24.4	48.8
12	12.2	24.4
13	6.1	12.2
14	3.05	6.1

Duty Cycle

The PWMAx and PWMBx registers directly control the duty cycles of the two pairs of PWM output signals on the PWM_Ax to PWM_Bx pins when not in switch reluctance mode.

- The two's-complement integer value in the PWMAx registers controls the duty cycle of the signals on the PWM_AH and PWM_AL pins.
- The two's-complement integer value in the PWMBx registers control the duty cycle of the signals on PWM_BH and PWM_BL pins.

The duty cycle registers are programmed in two's-complement integer counts of the fundamental time unit, PCLK, and define the desired on-time of the high side PWM signal produced by the two-phase timing unit over half the PWM period. The duty cycle register range is from:

(–*PWMPERIOD* ÷ 2 – *PWMDT*) to (+*PWMPERIOD* ÷ 2 + *PWMDT*)

which, by definition, is scaled such that a value of 0 represents a 50% PWM duty cycle. The switching signals produced by the two-phase timing unit are also adjusted to incorporate the programmed dead time value in the PWMDT register. The two-phase timing unit produces active low signals so that a low level corresponds to a command to turn on the associated power device.

Output Enable

The PWMSEG register contains four bits (0 to 3) that can be used to individually enable or disable each of the 4 PWM outputs. If the associated bit of the PWMSEG register is set (=1), then the corresponding PWM output is disabled, regardless of the value of the corresponding duty cycle register. This PWM output signal remains disabled as long as the corresponding enable/disable bit of the PWMSEGx register is set. In single update mode, changes to this register only become effective at the start of each PWM

cycle. In double update mode, the PWMSEG register can also be updated at the mid-point of the PWM cycle.

After reset, all four enable bits of the PWMSEG register are cleared so that all PWM outputs are enabled by default.

Crossover Mode

The PWMSEG3-0 registers contain two bits (PWM_AXOV and PWM_BXOV), one for each PWM output (see "PWM Output Disable Registers (PWMSEGx)" on page A-27). If crossover mode is enabled for any pair of PWM signals, the high-side PWM signal from the timing unit (for example, AH) is diverted to the associated low side output of the output control unit so that the signal ultimately appears at the AL pin.

The corresponding low side output of the timing unit is also diverted to the complementary high side output of the output control unit so that the signal appears at the AH pin. Following a reset, the two crossover bits are cleared so that the crossover mode is disabled on both pairs of PWM signals. Even though crossover is considered an output control feature, dead time insertion occurs after crossover transitions to eliminate shoot-through safety issues.

Note that crossover mode does not work if:

1. One signal of PWM_AL–PWM_AH or PWM_BL–PWM_BH is disabled.
2. PWM_AL and PWM_AH or PWM_BL and PWM_BH have different polarity settings from PWMPOLx registers.

In other words, both PWM_AL and PWM_AH or PWM_BL and PWM_BH should be enabled and both should have same polarity for proper operation of cross-over mode.

Interrupts

For interrupt execution, the specific PWM_IRQEN bit in the corresponding PWMCTLx register must be set including the IMASK or LIRPTL registers based on the programmable interrupt to be used.

Whenever a period starts, the PWM interrupt is generated. Since all four PWM units share the same interrupt vector, the interrupt service routine should read the PWMGSTAT register in order to determine the source of the interrupt. Next, the ISR clears the status bits of the PWMGSTAT register by explicitly writing 1.

Debug Features

The module contains four debug status registers (PWMDBG3-0), which can be used for debug aid. Each register is available per unit. The registers return current status information about the AH, AL, BH, BL output pins.

Programming Example

Listing 10-1 shows the four steps used to configure a PWM module.

Listing 10-1. Generic PWM Configuration Example

```
#include <def21364.h>
#include <sru21364.h>
#include <SRU.h>

/* define for PWM frequency used in PWMPERIOD0 */
#define fPWM 0x1388;          /* 200MHz/2(20kHz) => 50us */

call Int_enable;
```

```
call PWM_setup;
call PWM_enable;
nop;
finish: jump finish;


/* enable interrupts */
Int_enable:
LIRPTL = 0;
bit set MODE1 IRPTEN;         /* Global interrupt enable */
bit set LIRPTL P13IMSK;       /* Enable PWM default interrupt
                                 location 13 */
Int_enable.end: rts;

/* PWM setup registers */
PWM_setup:

/* 1. Configure frequency */
ustat3=fPWM;              /* fCK/2xfPWM */
dm(PWMPERIOD0)=ustat3;    /* PWM Period Register for switching
                             frequency (unsigned integer) */

/* 2. Configure duty cycles Width=[period/2] + duty program in
the 2's compliment of the high side width for individual control
this only programs AH signal. If PWMAL0 is not programmed then
the AL and AH signals have the same duty cycle */

ustat3=0;                 /* PWM Channel A Duty Control
                             (2s compliment integer) */
dm(PWMA0)=ustat3;         /* Set up the duty cycle register to 0
                             (50% duty cycle) */
ustat3 = 0x63C;           /* 2 compliment of 0x9C4 = 0x63C - 80%
                             high - 20% low */
dm(PWMAL0)=ustat3;        /* PWM Channel AL Duty Control */
```

```
/* 3. Configure Dead Time */
ustat3=0x0;                /* PWM Dead Time Register (unsigned
                             integer) */
dm(PWMDT0)=ustat3;

/* 4. Configure Polarity (this can be changed on the fly
after the PWM port is enabled) */

ustat3=0;                  /* PWM Polarity Select Register */
bit set ustat3 PWM_POL1AL | PWM_POL1AH;     /* Enables high
                                            polarityA output */
dm(PWMPOL0)=ustat3;
PWM_setup.end: rts;

/* PWM enable */
PWM_enable:
ustat3=dm(SYSCTL);           /* System Control Register */
bit set ustat3 PWM0EN | PPFLGS;
dm(SYSCTL)=ustat3;             /* Selects AD[11:8] in PWM0 mode
                                 instead of PP mode */
ustat3=dm(PWMSEG0);          /* PWM Output Enable */
bit set ustat3 PWM_BH | PWM_BL;   /* disables B outputs */
dm(PWMSEG0)=ustat3;
ustat3=dm(PWMCTL0);          /* PWM0 Control Register */
bit set ustat3 PWM_IRQEN;    /* enable PWM0 interrupt */
dm(PWMCTL0)=ustat3;
ustat3=dm(PWMGCTL);          /* PWM General Control Register */
bit set ustat3 PWM_EN0 | PWM_DIS1 | PWM_DIS2 | PWM_DIS3 |
PWM_SYNCEN0 | PWM_SYNCDIS1| PWM_SYNCDIS2| PWM_SYNCDIS3;
dm(PWMGCTL)=ustat3;

/* Enables only PWM 0 and it's internal timer; Disables other
```

```
PWMs globally. The write to PWMGCTL will kick off the transfer */

PWM_enable.end: rts;
```

11 SONY/PHILIPS DIGITAL INTERFACE

The Sony/Philips Digital Interface (S/PDIF) is a standard audio data transfer format that allows the transfer of digital audio signals from one device to another without having to convert them to an analog signal. Its primary features are listed in Table 11-1.

Table 11-1. S/PDIF Feature Summary

Feature	Transmitter	Receiver
Connectivity		
Multiplexed Pinout	No	No
SRU DAI Required	Yes	Yes
SRU DAI Default Routing	No	No
Interrupt Default Routing	Yes (P0I, P12I)	Yes (P0I, P12I)
Protocol		
Master Capable	No	Yes
Slave Capable	Yes	No
Transmission Simplex	Yes	Yes
Transmission Half Duplex	No	No
Transmission Full Duplex	No	No
Access Type		
Data Buffer	No	No
Core Data Access	N/A	N/A
DMA Data Access	N/A	N/A
DMA Channels	N/A	N/A

Table 11-1. S/PDIF Feature Summary (Cont'd)

Feature	Transmitter	Receiver
DMA Chaining	N/A	N/A
Interrupt Source	Core/(DAI)	Core (DAI)
Boot Capable	N/A	N/A
Local Memory	No	No
Clock Operation	PCLK/4	PCLK/4

Features

The S/PDIF interface has the following additional features.

- AES3-compliant S/PDIF transmitter and receiver.
- Transmitting a biphase mark encoded signal that may contain any number of audio channels (compressed or linear PCM) or non-audio data.
- S/PDIF receiver managing clock recovery with separate S/PDIF PLL or optional using external PLL circuit
- S/PDIF receiver direct supports DTS frames of 256, 512 and 1024
- Managing user status information and providing error-handling capabilities in both the transmitter and receiver.
- DAI allows interactions over DAI by serial ports, IDP and/or the external DAI pins to interface to other S/PDIF devices.

  This includes using the receiver to decode incoming biphase encoded audio streams and passing them via the SPORTs to internal memory for processing-or using the transmitter to encode audio or digital data and transfer it to another S/PDIF receiver in the audio system.

Notice it is important to be familiar with serial digital application interface standards IEC-60958, EIAJ CP-340, AES3 and AES11.

S/PDIF Transmitter

The following sections provide information on the S/PDIF transmitter.

Pin Descriptions

Table 11-2 provides descriptions of the pins used for the S/PDIF transmitter.

Table 11-2. S/PDIF Transmitter Pin Descriptions

Internal Node	I/O	Description
DIT_CLK_I	Input	**Serial clock.** Controls the rate at which serial data enters the S/PDIF module. This is typically 64 time slots.[1]
DIT_DAT_I	Input	**Serial Data.** The format of the serial data can be I^2S, and right- or left-justified.
DIT_FS_I	Input	**Serial Frame Sync.**
DIT_HFCLK_I	Input	**Input sampling clock.** The over sampling clock (which is divided down according to the FREQMULT bit in the transmitter control register to generate the biphase clock)
DIT_EXTSYNC_I	Input	**External synchronization.** Used for synchronizing the fame counter. If external synchronization is enabled (bit 15 of DITCTL is set), frame counter resets at rising edge of LRCLK next to the rising edge of EXT_SYNC_I.

Table 11-2. S/PDIF Transmitter Pin Descriptions (Cont'd)

Internal Node	I/O	Description
DIT_O	Output	**Transmit biphase mark encoded data stream.**
DIT_BLKSTART_O	Output	**Transmit block start.** Indicates the last frame of the current block. This is high for the entire duration of the last frame. This output can only be routed to the DAI interrupt 26 (SRU_EXT_MISCB register) for interrupts.

1 Timing for the S/PDIF format consists of time slots, unit intervals, subframes, and frames. For a complete explanation of S/PDIF timing, see one of the digital application interface standards listed in the "Features" section of this chapter.

SRU Programming

The SRU (signal routing unit) is used to connect the S/PDIF transmitter biphase data out to the output pins or to the S/PDIF receiver. The serial clock, frame sync, data, and EXT_SYNC (if external synchronization is required) inputs also need to be routed through SRU (see Table 11-3).

Table 11-3. S/PDIF DAI/SRU Transmitter Signal Connections

Internal Node	DAI Group	SRU Register
Inputs		
DIT_CLK_I DIT_HFCLK_I DIT_EXTSYNC_I	Group A	SRU_CLK4–2
DIT_DAT_I	Group B	SRU_DAT4
DIT_FS_I	Group C	SRU_FS2
Outputs		
DIT_O	Group C, D	
DIT_BLKSTART_O	Group E	

Functional Description

The S/PDIF transmitter, shown in Figure 11-1 resides within the DAI, and it's inputs and outputs can be routed via the SRU. It receives audio data in serial format, encloses the specified user status information, and converts it into the biphase encoded signal. The serial data input to the transmitter can be formatted as left-justified, I^2S, or right-justified with word widths of 16, 18, 20 or 24 bits. Figure 11-2 shows detail of the AES block.

The serial data, clock, and frame sync inputs to the S/PDIF transmitter are routed through the signal routing unit (SRU). For more information, see "DAI Signal Routing Unit Registers" on page A-80.

The S/PDIF transmitter output may be routed to an output pin via the SRU and then routed to another S/PDIF receiver or to components for off-board connections to other S/PDIF receivers. The output is also available to the S/PDIF receiver for loop-back testing through SRU.

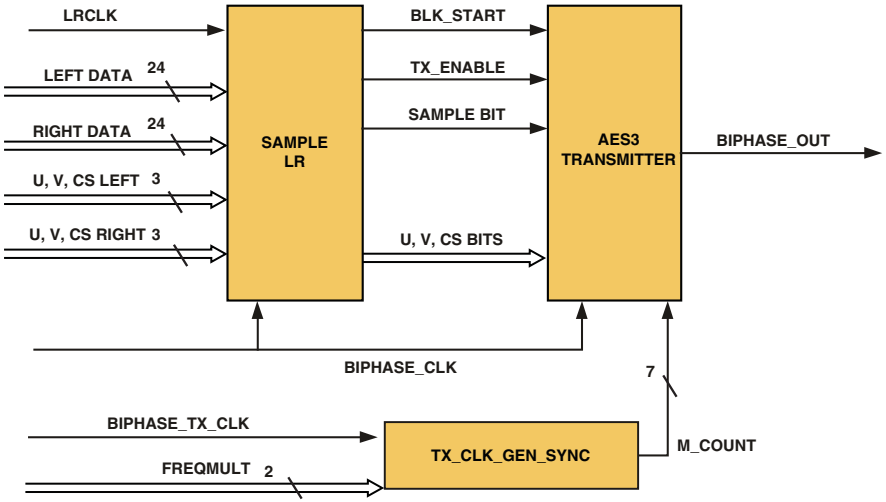


Figure 11-1. S/PDIF Transmitter Block Diagram

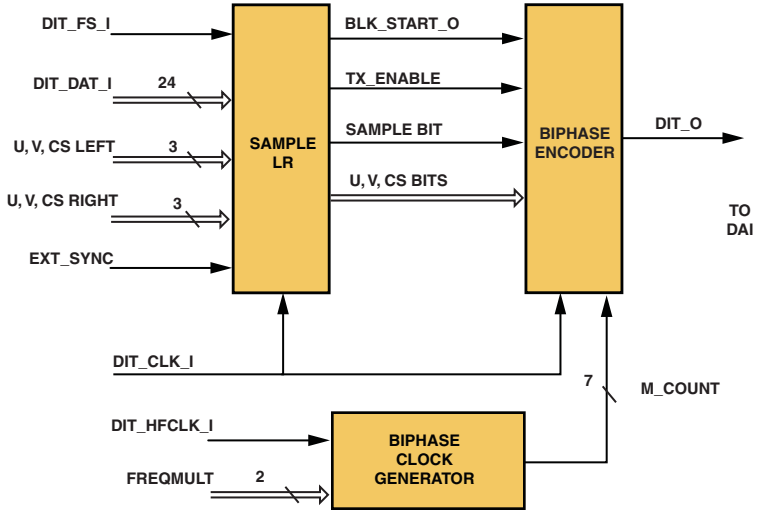


Figure 11-2. AES3 Output Block

In addition to encoding the audio data in the bi-phase format, the transmitter also provides a way to easily add the channel status information to the outgoing bi-phase stream. There are status registers for a frame (192-bits/24 bytes) in the transmitter that correspond to each channel or subframe. For more information, see "Transmit Control Register (DITCTL)" on page A-70.

Validity bits for both channels may also be controlled by the transmitter control register. Optionally, the user bit, validity bit, and channel status bit are sent to the transmitter with each left/right sample. For each subframe the parity bit is automatically generated and inserted into the bi-phase encoded data. A mute control and support for double-frequency single-channel mode are also provided. The serial data input format may be selected as left-justified, I^2S, or right-justified with 16-, 18-, 20- or 24-bit word widths. The over sampling clock is also selected by the transmitter control register.

Input Data Format

The Figure 11-3 through Figure 11-7 shows the format of data that is sent to the S/PDIF transmitter using a variety of interfaces.

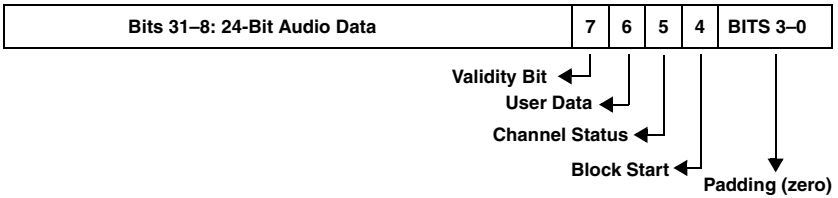


Figure 11-3. Data Packing for I²S and Left-Justified Format

When I²S format is used with 20-bit or 16-bit data, the audio data should be placed from the MSB of the 24-bit audio data.

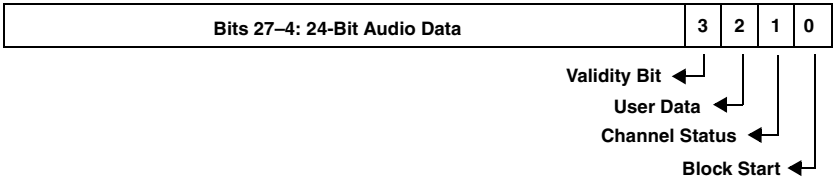


Figure 11-4. Data Packing for Right-Justified Format, 24 Bits

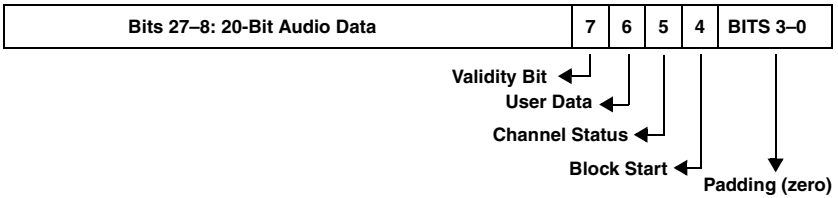


Figure 11-5. Data Packing for Right-Justified Format, 20 Bits

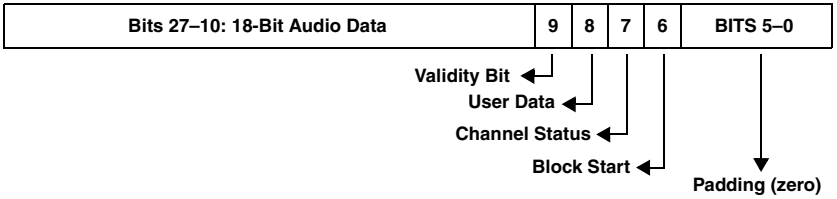


Figure 11-6. Data Packing for Right-Justified Format, 18 Bits

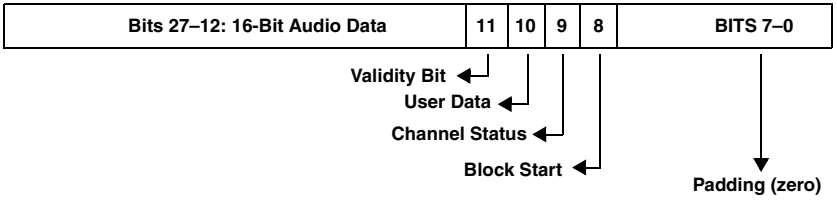


Figure 11-7. Data Packing for Right-Justified Format, 16 Bits

Output Data Mode

Two output data formats are supported by the transmitter; *two channel mode* and *single-channel double-frequency* (SCDF) mode. The output format is determined by the transmitter control register (DITCTL).

In two channel mode, the left channel(channel A) is transmitted when the DIT_FS_I is high and the right channel (channel B) is transmitted when the DIT_FS_I is low.

In SCDF mode, the transmitter sends successive audio samples of the same signal across both sub frames, instead of channel A and B. The transmitter will transmit at half the sample rate of the input bit stream. The DIT_SCDF bit (bit 4 in the DITCTL register selects SCDF mode. When in SCDF mode, the DIT_SCDF_LR bit (bit 5 in the DITCTL register) register decides whether left or right channel data is transmitted. For more information, see “Transmit Control Register (DITCTL)” on page A-70.

Operation Modes

The S/PDIF transmitter can operate in standalone and full serial modes. The following sections describe these modes in detail.

Standalone Mode

This mode is selected by setting bit 9 in the DITCTL register. In this mode, the block start bit (indicating the start of a frame) is generated internally. The channel status bits come from the channel status buffer registers (DITCHANAx and DITCHANBx). The channel status buffer must be programmed before the S/PDIF transmitter is enabled and used for all the successive blocks of data.

The validity bit for channel A and B are taken from bit 10 and bit 11 of the DITCTL register. In this mode only audio data comes from the DIT_DATA_I pin. All other data, including the status bit and block start bit is either generated internally or taken from the internal register.

Full Serial Mode

This mode is selected by clearing bit 9 in the DITCTL register. In this mode all the status bits, audio data and the block start bit (indicating start of a frame), come through the SDATA pin. The transmitter should be enabled after or at the same time as all of the other control bits.

Register Descriptions

The following sections describe the registers that control data transmit functions.

Control Register (DITCTL)

The DITCTL register contains control parameters for the S/PDIF transmitter. The control parameters include transmitter enable, mute information,

over sampling clock division ratio, SCDF mode select and enable, serial data input format select and validity and channel status buffer selects. By default, all the bits in this register are zero.

If the channel status or validity buffer needs to be enabled (after the SRU programming is complete), first write to the buffers with the required data and then enable the buffers by setting DIT_AUTO (bit 9 of DITCTL register). Setting DIT_AUTO bit also results in the block start bit (indicating start of a frame) being generated internally. Also use this register to write other control values such as DIT_SMODEIN, DIT_FREQ, and enable the transmitter by setting the DIT_EN bit.

Channel Status Registers (DITCHANAx/Bx)

These registers provide status information for transmitter subframe A and B. The first five bytes of the channel status may be written all at once to the control registers for both A and B channels. As the data is serialized and transmitted, the appropriate bit is inserted into the channel status area of the 192-word frame. Note these registers are used in standalone mode only.

S/PDIF Receiver

The S/PDIF receiver (Figure 11-8) is compliant with all common serial digital application interface standards including IEC-60958, IEC-61937, AES3, and AES11. These standards define a group of protocols that are commonly associated with the S/PDIF interface standard defined by AES3, which was developed and is maintained by the Audio Engineering Society. The AES3 standard effectively defines the data and status bit structure of an S/PDIF stream. AES3-compliant data is sometimes referred to as AES/EBU compliant. This term highlights the adoption of the AES3 standard by the European Broadcasting Union.

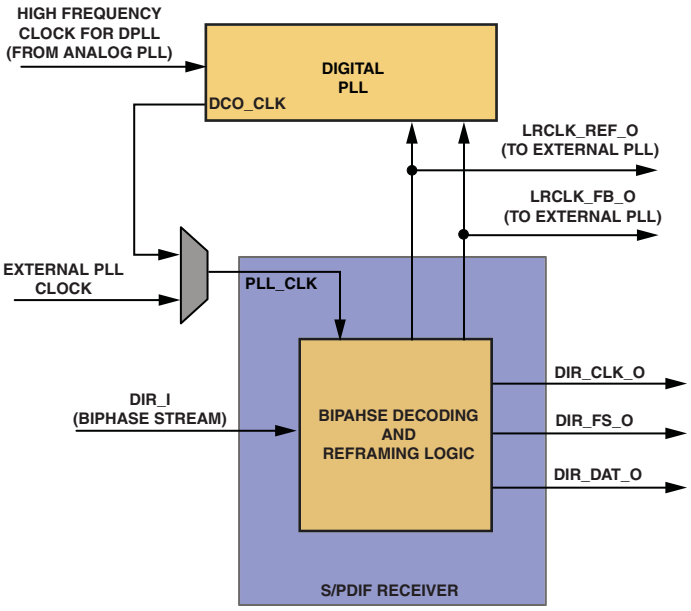


Figure 11-8. S/PDIF Receiver Block Diagram

Pin Descriptions

Table 11-4 provides descriptions of the pins used for the S/PDIF receiver.

Table 11-4. S/PDIF Receiver Pin Descriptions

Internal Node	I/O	Description
SPDIF_EXTPLLCLK_I	Input	PLL clock input (512 × FS). Input clock for external PLL.
DIR_I	Input	Biphase mark encoded data receiver input stream.
DIR_CLK_O	Output	Extracted receiver sample clock output.
DIR_TDMCLK_O	Output	Receiver TDM clock out. This clock is 4 times DIR_CLK_O.
DIR_FS_O	Output	Extracted receiver frame sync out.

Table 11-4. S/PDIF Receiver Pin Descriptions (Cont'd)

Internal Node	I/O	Description
DIR_DAT_O	Output	Extracted audio data output.
DIR_LRCLK_FB_O	Output	Receiver frame sync feed back output. Input for external PLL.
DIR_LRCLK_REF_O	Output	Receiver frame sync reference clock output. Input for external PLL.

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the S/PDIF receiver to the output pins or any other peripherals and also for the connection to the input biphase stream.

Program the corresponding SRU registers to connect the outputs to the required destinations (Table 11-5). The biphase encoded data and the external PLL clock inputs to the receiver are routed through the signal routing unit (SRU). The extracted clock, frame sync, and data are also routed through the SRU.

Table 11-5. S/PDIF DAI/SRU Receiver Signal Connections

Internal Node	DAI Group	SRU Register
Inputs		
SPDIF_EXTPLLCLK_I	Group A	SRU_CLK4
DIR_I	Group C	SRU_FS3
Outputs		
DIR_CLK_O DIR_TDMCLK_O	Group A, D	
DIR_FS_O	Group C, D	
DIR_DAT_O	Group B, D	
DIR_LRCLK_FB_O DIR_LRCLK_REF_O	Group D	

Functional Description

The input to the receiver (DIR_I) is a biphase encoded signal that may contain two audio channels (compressed or linear PCM) or non-audio data. The receiver decodes the single biphase encoded stream, producing an I^2S compatible serial data output that consists of a serial clock, a left-right frame sync, and data (channel A/B). It provides the programmer with several methods of managing the incoming status bit information.

The S/PDIF receiver receives any S/PDIF stream with a sampling frequency range of 32 kHz – 15% to 192 kHz + 15% range.

The channel status bits are collected into memory-mapped registers, while other channel status and user bytes must be handled manually. The block start bit, which replaces the parity bit in the serial I^2S stream, indicates the reception of the Z preamble and the start of a new block of channel status and data bits.

Output Data Format

The extracted 24-bit audio data, V, U, C and block start bits are sent on the DIR_DAT_O pin in 32-bit I^2S format as shown in Figure 11-3. The frame sync is transmitted on the DIR_FS_O pin and serial clock is transmitted on the DIR_CLK_O pin. All three pins are routed through the SRU.

PLL Selection for Clock Recovery

The S/PDIF receiver extracts audio data, channel status, and user bits from the biphase encoded AES3 and S/PDIF stream. In addition, a 50% duty cycle reference clock running at the sampling rate of the audio input data is generated for the PLL in the receiver to recover the oversampling clock.

The receiver can recover the clock from the biphase encoded stream using an on-chip digital PLL (the dedicated on-chip digital PLL is separate from the digital PLL that supplies the SHARC processor core).

The left/right frame reference clock for the PLL is generated using the preambles. The recovered low jitter left/right frame clock from the PLL attempts to align with the reference clock. However, this recovered left/right clock, like the reference clock, is not phase aligned with the preambles.

Notice there are various performance characteristics to consider when configuring for analog PLL mode. For more information using PLLs, visit the Analog Devices Inc. web site at:
`http://www.analog.com/embedded-processing-dsp/processors`

Register Descriptions

The ADSP-2136x processor contains four registers that are used to enable/disable S/PDIF receiver, to manage its operation, and to report status. The registers are as follows.

Receiver Control Register (DIRCTL)

The `DIRCTL` register contains control parameters for the S/PDIF receiver. The control parameters include mute information, error controls, SCDF mode select and enable, and S/PDIF PLL disable.

The S/PDIF receiver is enabled at default to receive in two-channel mode.

If the receiver is not used, programs should disable the digital PLL to avoid unnecessary switching. This is accomplished by writing into the `DIR_PLLDIS` bit (bit 7) in the `DIRCTL` register. In most cases, when the S/PDIF receiver is used, this register does not need to be changed. After the SRU programming is complete, write to the `DIRCTL` register with con-

trol values, and enable the internal digital PLL by clearing the DIR_PLLDIS bit if it was cleared initially. At this point, the receiver attempts to lock.

For a detailed description of this register, see "Receive Control Register (DIRCTL)" on page A-73.

Receiver Status Register (DIRSTAT)

The receiver also detects errors in the S/PDIF stream. These error bits are stored in the status register, which can be read by the core. Optionally, an interrupt may be generated to notify the core on error conditions.

This 32-bit read-only register is used to store the error bits. The error bits are sticky on read. Once they are set, they remain set until the register is read. This register also contains the lower byte of the 40-bit channel status information.

Lock error. When bit 4 in the DIRSTAT register is set (=1), the PLL is locked. When the DIR_LOCK bit in the DIRSTAT register is deasserted, it means the PLL has become unlocked and the audio data is handled according to the DIR_LOCK_ERR (bits 3–2 in DIRCTL register). When this happens, the receiver functions as follows.

- 00 = No action is taken with the audio data.
- 01 = The last valid audio sample is held.
- 10 = Zeros are sent out after the last valid sample.
- 11 = Soft mute of the last valid audio sample is performed (as if DIR_NOSTREAM is asserted).

This is valid only when linear PCM audio data is in the stream. When non-linear audio data is in the stream, this mode defaults to the case of DIR_LOCK_ERR1-0 bits = 10.

Once the receiver is locked, the corresponding DIR_LOCK bit in the DIRSTAT register is set. This bit can be polled to detect the DIR_LOCK condi-

tion. After the receiver is locked, the other status bits in the receiver status (DIRSTAT) and the channel status (DIRCHANL/R) registers can be read. Interrupts can be also used with some status bits.

No Stream error. The DIR_NOSTREAM bit (5) is asserted whenever the AES3/SPDIF stream is disconnected. When the DIR_NOSTREAM bit is asserted and the audio data in the stream is linear PCM, the receiver performs a soft mute of the last validsample from the AES3/SPDIF stream. A soft mute consists of taking the last valid audio sample and slowly and linearly decrementing it to zero, over a period of 4096 frames. During this time, the PLL three-states the charge pump until the soft mute has been completed. If non-linear PCM audio data is in the AES3/SPDIF stream when the NOSTREAM bit is asserted, the receiversends out zeros after the last valid sample.

Parity error. When bit 6 in the DIRSTAT register is set (=1), it indicates that the AES3/SPDIF stream was received with the correct parity, or even parity. When the DIR_PARITYERROR bit is low (=0), it indicates that an error has occurred, and the parity is odd. When a parity or biphase error occurs, the audio data is handled according to the DIR_BIPHASE1-0 bits in DIRCTL the following manner.

- 00 = No action is taken with the audio data.
- 01 = The last valid sample is held.
- 10 = The invalid sample is replaced with zeros.

Biphase error. When bit 7 in the DIRSTAT register is set (=1), it indicates that a biphase error (DIR_BIPHASEERROR) has occurred and the data sampled from the biphase stream may not be correct.

The VALIDITY, NONAUDIO, NOSTREAM, BIPHERR, PARITY and LOCK bits are also stored in the receiver status register as sticky bits.

Clock Recovery

The phased-locked loop for the AES3/SPDIF receiver is intended to recover the clock that generated the AES3/SPDIF biphase encoded stream. This clock is used by the receiver to clock in the biphase encoded data stream and also to provide clocks for either the SPORTs, sample rate converter, or the AES3/SPDIF transmitter. The recovered clock may also be used externally to the chip for clocking D/A and A/D converters.

To be AES11 compliant, the recovered left/right clock must be aligned with the preambles within ±5% of the frame period. Since the PLL generates a clock 512 timesthe frame rate clock (512× FSCLK), this clock can be used and divided down to create the phase aligned jitter-free left/right clock. For more information on recovered clocks, see "PLL Selection for Clock Recovery" on page 11-13.

Note that jitter on the recovered clock must be less than 200 ps and, if possible, less than 100 ps across all the sampling frequencies ranging from 27.2 kHz to 220.8 kHz (32 kHz – 15% and 192 kHz + 15%). Furthermore, once the PLL achieves lock it should be able to vary ±15% in frequency over time. This allows for applications that do not use PLL unlocking.

Channel Decoding

This section describes the receiver channel status for the different modes.

Channel Status

The channel status for the first bytes 4–0 are collected into memory-mapped registers (DIRCTL and DIRCHANA/DIRCHANB registers). All other channel status bytes 23–5 must be manually extracted from the receiver data stream.

Only the first 5 channel status bytes (40-bit) of a frame are stored into the S/PDIF receiver status registers.

Compressed or Non-linear Audio Data

The S/PDIF receiver processes compressed as well as non-linear audio data according to the supported standards. The following sections describe how this peripheral handles different data.

The AES3/SPDIF receiver is required to detect compressed or non-linear audio data according to the AES3, IEC60958, and IEC61937 standards. Bit 1 of byte 0 in the DIRSTAT register indicates whether the audio data is linear PCM, (bit 1=0), or non-PCM audio, (bit 1=1). If the channel status indicates non-PCM audio, the DIR_NOAUDIO bit flag is set. (This bit can be used to generate an interrupt.) The DIR_VALID bit (bit 3 in the DIRSTAT register) when set (=1) may indicate non-linear audio data as well. Whenever this bit is set, the VALIDITY bit flag is set in the DIR_STAT register.

MPEG-2, AC-3, DTS, and AAC compressed data may be transmitted without setting either the DIR_VALID bit or bit 1 of byte 0. To detect this data, the IEC61937 and SPMTE 337M standards dictate that there be a 96-bit sync code in the 16-, 20- or 24-bit audio data stream. This sync code consists of four words of zeros followed by a word consisting of 0xF872 and another word consisting of 0x4E1F. When this sync code is detected, the DIR_NOAUDIO bit flag is set. If the sync code is not detected again within 4096 frames, the DIR_NOAUDIO bit flag is deasserted.

The last two words of the sync code, 0xF872 and 0x4E1F, are called the preamble-A and preamble-B of the burst preamble. Preamble-C of the burst preamble contains burst information and is captured and stored by the receiver. Preamble-D of the burst preamble contains the length code and is captured by the receiver. Even if the validity bit or bit 1 of byte 0 has been set, the receiver still looks for the sync code in order to record the preamble-C and D values. Once the sync code has not been detected in 4096 frames, the preamble-C and D registers are set to zero.

The S/PDIF receiver supports the DTS stream. The DTS specifications support frame sizes of 256, 512, 1024, 2048 and 4096. The on-chip S/PDIF receiver supports the 256, 512 and 1024 DTS frames. The DTS test kit frames with 2048 and 4096 frame sizes can be detected by adding the sync detection logic in software by using a software counter to check for the DTS header every 2048 and 4096 frames respectively.

Emphasized Audio Data

The receiver must indicate to the program whether the received audio data is emphasized using the channel status bits as detailed below.

- In professional mode, (bit 0 of byte 0 = 1), channel status bits 2–4 of byte 0 indicate the audio data is emphasized if they are equal to 110 or 111.
- In consumer mode, (bit 0 of byte 0 = 0), channel status bits 3–5 indicate the audio data is emphasized if they are equal to 100, 010 or 110.

If emphasis is indicated in the channel status bits, the receiver asserts the EMPHASIS bit flag. This bit flag is used to generate an interrupt.

Single-Channel Double-Frequency Mode

Single-channel, double-frequency mode (SCDF) mode is selected with DIR_SCDF and DIR_SCDF_LR bits in the DIRCTL register. The DIR_BOCHANL/R bits in the DIRSTAT register also contain information about the SCDF mode. When the DIR_BOCHANL/R indicates single channel double frequency mode, the two subframes of a frame carry successive audio samples of the same signal. Bits 0–3 of channel status byte 1 are decoded by the receiver to determine one of the following:

- 0111 = single channel double frequency mode

- 1000 = single channel double frequency mode–stereo left
- 1001 = single channel double frequency mode–stereo right

Interrupts

All S/PDIF interrupts are generated by the transmitter and receiver and processed through the DAI interrupt controller which can generate an interrupt signal using the (DAI_IRPTL_x) registers.

Transmitter Interrupt

The DIT_BLKSTART_O output signal, if routed to miscellaneous interrupt bit 26 only (DAI_INT_26_I in SRU_EXT_MISCB register), triggers a block start interrupt during the last frame of current block.

Receiver Interrupts

The following eight receiver status bits can generate an interrupt.

- DIR_LOCK_INT
- DIR_VALID_INT
- DIR_NOSTREAM_INT
- DIR_NOAUDIO_INT
- DIR_CRCERROR_INT
- DIR_EMPHASIS_INT
- DIR_ERROR_INT
- DIR_STATCNG_INT

Notice that parity error and biphase error are ORed together to form a DIR_ERROR_INT interrupt.

Whenever there is a change in channel status information, the DIR_STATCNG_INT bit in the DAI_IRPTL_x register is set. The DIR_CRCERROR_INT bit is asserted high whenever the CRCC check of the DIR_BOCHANL/R bits fail. The CRCC check isonly performed if the channel status bit 0 of byte 0 is high, indicating professional mode. If emphasis is indicated in the channel statusbits, the receiver sets theDIR_EMPHASIS_INT bit in the DAI_IRPTL_x register.

Debug Features

The following section provides information on loopback routing which can be used in debugging this peripheral.

Loopback Routing

The S/PDIF supports an internal loopback mode by using the SRU. For more information, see “Loop Back Routing” on page 5-30.

Programming Model

The following sections provide information on programming the transmitter and receiver.

Programming the Transmitter

Since the S/PDIF transmitter data input is not available to the core, programming the transmitter is as simple as: 1) connecting the SRU to the on-chip (serial ports or input data port) or off-chip (DAI pins) serial devices that provide the clock and data to be encoded, and 2) selecting the

desired mode in the transmitter control register. This setup can be accomplished in three steps.

1. Connect the transmitter's four required input signals and one biphase encoded output in the SRU. The four input signals are the serial clock (DIT_CLK_I), the serial frame sync (DIT_FS_I), the serial data (DIT_DAT_I), and the high frequency clock (DIT_HFCLK_I) used for the encoding. The only output of the transmitter is DIT_O.
2. Also route the DIT_BLK_START_O signal to the DAI_INT_26 (DAI_IRPTLx register). This generates interrupts during the last frame of the block (192), allowing changes of user bits for the next block.
3. Initialize the DITCTL register to enable the data encoding.
4. Manually set the block start bit in the data stream once per block (384 words). This is needed if automatic generation of block start information is not enabled in the DITCTL register, (DIT_AUTO = 0).

Programming the Receiver

Since the S/PDIF receiver data output is not available to the core, programming the peripheral is as simple as connecting the SRU to the on-chip (serial ports or input data port) or off-chip (DAI pins) serial devices that provide the clock and data to be decoded, and selecting the desired mode in the receiver control register. This setup can be accomplished in two steps.

1. Connect the input signal and three output signals in the SRU for. The only input of the receiver is the biphase encoded stream, DIR_I. The three required output signals are the serial clock (DIR_CLK_O), the serial frame sync (DIR_FS_O), and the serial data (DIR_DAT_O). The high frequency clock (DIR_TDMCLK_O) derived from the encoded stream is also available if the system requires it.

2. Initialize the DIRCTL register to enable the data decoding. Note that this peripheral is enabled by default.

Interrupted Data Streams on the Receiver

When using the S/PDIF receiver with data streams that are likely to be interrupted, (in other words unplugged and reconnected), it is necessary to take some extra steps to ensure that the S/PDIF receiver's digital PLL will re lock to the stream. The steps to accomplish this are described below.

1. Setup interrupts within the DAI so that the S/PDIF receiver can generate an interrupt when the stream is reconnected.
2. Within the interrupt service routine (ISR), stop and restart the digital PLL. This is accomplished by setting and then clearing bit 7 of the S/PDIF receiver control register.
3. Return from the ISR and continue normal operation.

This method of resetting the digital PLL has been shown to provide extremely reliable performance when S/PDIF inputs that are interrupted or unplugged momentarily occur.

The following procedure and the example code show how to reset the digital PLL. Note that all of the S/PDIF receiver interrupts are handled through the DAI interrupt controller. For more information, see "Functional Description" on page 5-3.

1. Initialize the No Stream Interrupt

```
/* Enable interrupts (globally) */
BIT SET MODE1 IRPTEN;
/* unmask DAI Hi=Priority Interrupt */
bit set imask DAIHI;
ustat1 = DIR_NOSTREAM_INT;
```

```
/* Enable no-stream Interrupt on Falling Edge. Interrupt
occurs when the stream is reconnected */
dm(DAI_IRPTL_FE) = ustat1;

/* Enable Hi-priority DAI interrupt */
dm(DAI_IRPTL_PRI) = ustat1;

/* If more than 1 DAI interrupt is being used, it is neces-
sary to determine which interrupt occurred here */

/* Interrupt Service Routine for the DAI Hi-Priority Inter-
rupt. This ISR triggered when the DIR sets no_stream bit */
_DAIisrH:
```

2. Reset the Digital PLL Inside of the ISR

```
r8=dm(DAI_IRPTL_H);          /* Reading DAI_IRPTL_H
                                clears interrupt */
ustat2=dm(DIRCTL);
 bit set ustat2 DIR_PLLDIS; /* bit_7 disables Dpll only */
 dm(DIRCTL)=ustat2;
 bit clr ustat2 DIR_PLLDIS;  /*reenable the digital pll */
 dm(DIRCTL)=ustat2;
```

12 ASYNCHRONOUS SAMPLE RATE CONVERTER

The asynchronous sample rate converter (SRC) block is used to perform synchronous or asynchronous sample rate conversion across independent stereo channels, without using any internal processor resources. Furthermore, the SRC blocks can be configured to operate together to convert multichannel audio data without phase mismatches. Finally, the SRC is used to clean up audio data from jittery clock sources such as the S/PDIF receiver.

Features

The SRC for the SHARC processors has the additional features shown in Table 12-1 and the list below.

Table 12-1. SRC Feature Summary

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	No
Interrupt Default Routing	Yes (P0I or P12I)
Protocol	
Master Capable	No
Slave Capable	Yes
Transmission Simplex	Yes

Table 12-1. SRC Feature Summary (Cont'd)

Feature	Availability
Transmission Half Duplex	No
Transmission Full Duplex	No
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Interrupt Source	Core (DAI)
Boot Capable	N/A
Local Memory	Yes (RAM, ROM)
Clock Operation	PCLK/4

- 4 sample rates converters
- Automatically senses sample frequencies
- Simple programming required
- Attenuates sample clock jitter
- Supports left-justified, I^2S, right-justified (16-,18-, 20-, 24-bits), and TDM serial port (daisy chain and matched phase) modes.
- Accepts 16-/18-/20-/24-bit data
- Up to 192 kHz sample rate input/output sample ratios from 7.75:1 to 1:8
- 140 or 128 dB SNR (depending on processor model)

- Matched Phase Mode to compensate for group delays (ADSP-21364 only)
- Linear phase FIR filter
- Controllable soft mute

Pin Descriptions

The SRC has two interfaces: an input port and an output port. Table 12-2 describes the six inputs and two outputs for the IP (input port) and OP (output port).

Table 12-2. SRC Pin Descriptions

Internal Node	I/O	Description
SRC3–0_CLK_IP_I	Input	SRC input port clock input
SRC3–0_FS_IP_I	Input	SRC input port frame sync input
SRC3–0_DAT_IP_I	Input	SRC input port data input
SRC3–0_CLK_OP_I	Input	SRC output port clock input
SRC3–0_FS_OP_I	Input	SRC output port frame sync input
SRC3–0_TDM_OP_I	Input	SRC output port TDM daisy chain data input
SRC3–0_DAT_OP_O	Output	SRC output port data output
SRC3–0_TDM_IP_O	Output	SRC output port TDM daisy chain data output

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the SRCs to the output pins or any other peripherals.

For normal operation, the data, clock, and frame sync signals need to be routed as shown in Table 12-3.

Table 12-3. SRC DAI/SRU Signal Routing

Internal Node	DAI Connection	SRU Register
Inputs		
SRC3–0_CLK_IP_I SRC3–0_CLK_OP_I	Group A	SRU_CLK2–1
SRC3–0_FS_IP_I SRC3–0_FS_OP_I	Group C	SRU_FS2–1
SRC3–0_DAT_IP_I SRC3–0_TDM_OP_I	Group B	SRU_DAT3–2
Outputs		
SRC3–0_DAT_OP_O	Group B, D	
SRC3–0_TDM_IP_O	Group B	

For information on using the SRU, see "Rules for SRU Connections" on page 5-16.

Functional Description

Figure 12-1 shows a top level block diagram of the SRC module and Figure 12-2 shows architecture details. The sample rate converter's FIFO block adjusts the left and right input samples and stores them for the FIR filter's convolution cycle. The SRCx_FS_IP counter provides the write address to the FIFO block and the ramp input to the digital-servo loop. The ROM stores the coefficients for the FIR filter convolution and performs a high-order interpolation between the stored coefficients. The sample rate ratio block measures the sample rate by dynamically altering the ROM coefficients and scaling the FIR filter length and input data. The digital-servo loop automatically tracks the `SRCx_FS_IP` and

SRCx_FS_OP sample rates and provides the RAM and ROM start addresses for the start of the FIR filter convolution.

Unlike other peripherals, the sample rate converters own local memories (RAM and ROM) which arededicated for the purpose of sample rate conversion only.

The master clock input (MCLK) shown in Figure 12-1 is peripheral clock (PCLK) divided by 4. Therefore, MCLK = PCLK ÷ 4.

Also note that the matched phase mode only applies to the ADSP-21364 sample rate converter. For all other SHARC processors, the SRC sends zeros at the LSB eight bits instead of the matched phase information when 32-bit clock is provided.

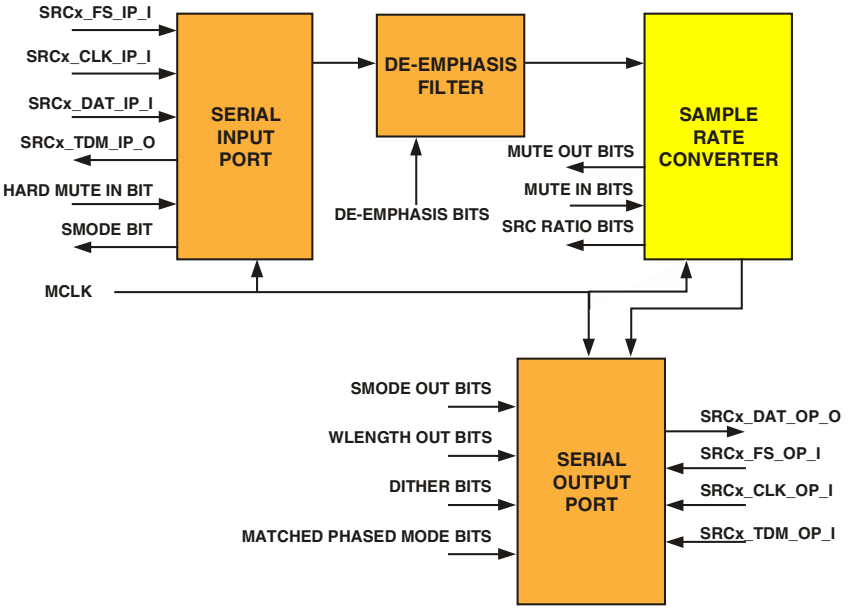


Figure 12-1. Sample Rate Converter Block Diagram

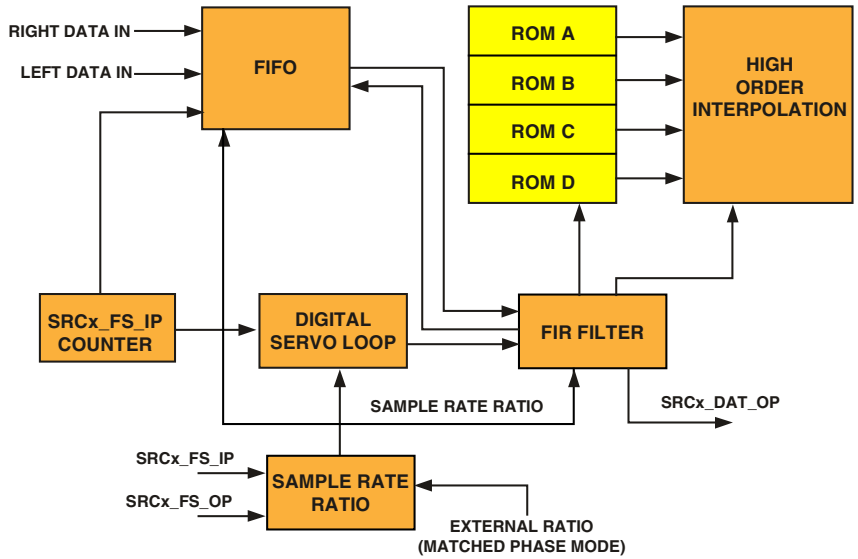


Figure 12-2. Sample Rate Converter Architecture

The FIFO receives the left and right input data and adjusts the amplitude of the data for both the soft muting of the SRC and the scaling of the input data by the sample rate ratio before storing the samples in RAM. The input data is scaled by the sample rate ratio because as the FIR filter length of the convolution increases, so does the amplitude of the convolution output. To keep the output of the FIR filter from saturating, the input data is scaled down by multiplying it by (SRCx_FS_OP)/(SRCx_FS_IP) when SRCx_FS_OP < SRCx_FS_IP. The FIFO also scales the input data to mute and stop muting the SRC.

The RAM in the FIFO is 512 words deep for both left and right channels. An offset of 64 to the write address, provided by the SRCx_FS_IP counter, is added to prevent the RAM read pointer from overlapping the write address. This offset value is useful for applications when small changes in the sample rate ratio between SRCx_FS_IP and SRCx_FS_OP are expected. The maximum decimation rate can be calculated from the RAM word depth is (512 – 64) ÷ 64 taps = 7.

The digital-servo loop is essentially a ramp filter that provides the initial pointer to the address in RAM and ROM for the start of the FIR convolution. The RAM pointer is the integer output of the ramp filter while the ROM pointer is the fractional part. The digital-servo loop must be able to provide excellent rejection of jitter on the SRCx_FS_IP and SRCx_FS_OP clocks as well as measure the arrival of the SRCx_FS_OP clock within 4.97 ps. The digital-servo loop also divides the fractional part of the ramp output by the ratio of (SRCx_FS_IP)/(SRCx_FS_OP) for the case when SRCx_FS_IP > SRCx_FS_OP, to dynamically alter the ROM coefficients.

The digital-servo loop is implemented with a multi-rate filter. To settle the digital-servo loop filter quickly at startup or at a change in the sample rate, a fast mode has been added to the filter. When the digital-servo loop starts up or the sample rate is changed, the digital-servo loop kicks into fast mode to adjust and settle on the new sample rate. Upon sensing the digital-servo loop settling down to some reasonable value, the digital-servo loop kicks into normal or slow mode. During fast mode, the SRCx_MUTE_OUT bit of the SRC is asserted to remind the user to mute the SRC which avoids clicks and pops.

The FIR filter is a 64-tap filter in the case of SRCx_FS_OP < SRCx_FS_IP and is (SRCx_FS_IP)/(SRCx_FS_OP) × 64 taps for the case when SRCx_FS_IP > SRCx_FS_OP. The FIR filter performs its convolution by loading in the starting address of the RAM address pointer and the ROM address pointer from the digital-servo loop at the start of the SRCx_FS_OP period. The FIR filter then steps through the RAM by decrementing its address by 1 for each tap, and the ROM pointer increments its address by the (SRCx_FS_OP/SRCx_FS_IP) × 2^{20} ratio for SRCx_FS_IP > SRCx_FS_OP or 2^{20} for SRCx_FS_OP < SRCx_FS_IP. Once the ROM address rolls over, the convolution is complete. The convolution is performed for both the left and right channels, and the multiply/accumulate circuit used for the convolution is shared between the channels.

The (SRCx_FS_IP)/(SRCx_FS_OP) sample rate ratio circuit is used to dynamically alter the coefficients in the ROM for the case when

SRCx_FS_IP > SRCx_FS_OP. The ratio is calculated by comparing the output of an SRCx_FS_OP counter to the output of an SRCx_FS_IP counter. If SRCx_FS_OP > SRCx_FS_IP, the ratio is held at one. If SRCx_FS_IP > SRCx_FS_OP, the sample rate ratio is updated if it is different by more than two SRCx_FS_OP periods from the previous SRCx_FS_OP to SRCx_FS_IP comparison. This is done to provide some hysteresis to prevent the filter length from oscillating and causing distortion.

However, the hysteresis of the (SRCx_FS_OP)/(SRCx_FS_IP) ratio circuit can cause phase mismatching between two SRCs operating with the same input and output clocks. Since the hysteresis requires a difference of more than two SRCx_FS_OP periods to update the SRCx_FS_OP and SRCx_FS_IP ratios, two SRCs may have differences in their ratios from 0 to 4 SRCx_FS_OP period counts. The (SRCx_FS_OP)/(SRCx_FS_IP) ratio adjusts the filter length of the SRC, which corresponds directly with the group delay. Thus, the magnitude in the phase difference depends upon the resolution of the SRCx_FS_OP and SRCx_FS_IP counters. The greater the resolution of the counters, the smaller the phase difference error.

Serial Data Ports

The serial data ports provide the interface through which data is transferred into and out of the SRC modules.

The SRC has a 3-wire interface for the serial input and output ports that supports left-justified, I^2S, and right-justified (16-, 18-, 20-, 24-bit) modes. Additionally, the serial interfaces support TDM mode for daisy-chaining multiple SRCs to a processor. The serial output data is dithered down to 20, 18, or 16 bits when 20-, 18-, or 16-bit output data is selected.

The SRC converts the data from the serial input port to the sample rate of the serial output port. The sample rate at the serial input port can be asynchronous with respect to the output sample rate of the output serial port.

De-Emphasis Filter

The SRCx_FS_IP_I signal asserts when a new frame of left and right data is available for the de-emphasis filter and the SRC. The de-emphasis filter is used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is selected by the SRCx_DEEMPHASIS1-0 bits and is based on the input sample rate as follows:

- 00 – No de-emphasis, audio data is passed directly to the SRC
- 01 – 32 kHz sample rate de-emphasis filter
- 10 – 44.1 kHz sample rate de-emphasis filter
- 11 – 48 kHz sample rate de-emphasis filter

Mute Control

When SRCx_ENABLE is enabled (set = 1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine.

- MUTE_OUT is asserted
- soft mute control counter for input samples is set to maximum attenuation (–144 dB)

When MUTE_OUT is asserted, the MUTE_IN signal should also be asserted to avoid any unwanted output.

SRC power-up completion is confirmed by clearing the SRCx_MUTEOUT bit in SRCRATx register.

Muting can also be controlled in software using the MUTE bits (SRCx_SOFTMUTE, SRCx_HARD_MUTE, SRCx_AUTO_MUTE) in the SRC control register (SRCCTL) as described below. For more information, see “Register Descriptions” on page 12-11.

Automatic Mute

The mute feature of the SRC can be controlled automatically in hardware using the MUTE_IN signal by connecting it to the MUTE_OUT signal. Note that by default, the SRCMUTE register connects the MUTE_IN signal to the MUTE_OUT signal, but not vice versa. Automatic muting can be disabled by setting (=1) the SRCx_MUTE_EN bits in the SRCMUTE register.

Soft Mute

When the SRCx_SOFTMUTE bit in the SRCCTL register is set, the MUTE_IN signal is asserted, and the SRC performs a soft mute by linearly decreasing the input data to the SRC FIFO to zero, (–144 dB) attenuation as described for automatic hardware muting.

A 12-bit counter, clocked by SRCx_FS_IP_I, is used to control the mute attenuation. Therefore, the time it takes from the assertion of MUTE_IN to –144 dB, full mute attenuation is 4096 FS cycles.

Likewise, the time it takes to reach 0 dB mute attenuation from the de-assertion of MUTE_IN is 4096 FS cycles.

Hard Mute

When the SRCx_HARD_MUTE bit in the SRCCTL register is set, the SRC immediately mutes the input data to the SRC FIFO to zero, (–144 dB) attenuation.

Auto Mute

When the SRCx_AUTO_MUTE bit in the SRCCTLx register is set, the SRC communicates with the S/PDIF receiver peripheral to determine when the input should mute. Each SRC is connected to the S/PDIF receiver to read the DIR_NOAUDIO bits (see"Receive Status Register (DIRSTAT)" on page A-75). When the DIR_NOAUDIO bit is set (=1), the SRC immediately mutes the input data to the SRC FIFO to zero, (–144 dB) attenuation.

This mode is useful for automatic detection of non-PCM audio data received from the S/PDIF receiver.

Register Descriptions

The SRC uses five registers to configure and operate the SRC module. For complete register and bit descriptions, see "Sample Rate Converter Registers" on page A-59.

Initially, programs configure the SRC control registers SRCCTL0 and SRCCTL1. The SRCCTL0 register contains control parameters for the SRC0 and SRC1 modules and the SRCCTL1 register contains control values for the SRC2 and SRC3 modules. The control parameters include mute information, data formats for input and output ports, de-emphasis enable, dither enable, and matched-phase mode enable (ADSP-21364 only) for multiple SRCs.

Write the settings to the desired control register at least one cycle before setting the corresponding SRC module enable bit, SRCx_ENABLE.

The following sections provide details on the SRC's control registers within the ADSP-2136x processors.

Control Registers (SRCCTLn)

When the SRCx_ENABLE bit (bit 31 in the SRCCTLx registers) is set (= 1), the SRC begins its initialization routine where all locations in the FIFO are initialized to zero, MUTE_OUT signal is asserted, and any output pins are enabled.

When setting and clearing the SRCx_ENABLE bit, it should be held low for a minimum of 2 PCLK cycles. It is recommended that the SRC be disabled when changing modes.

When the SRCx_ENABLE is set or there is a change in the sample rate between SRCx_FS_IP_I and SRCx_FS_OP_I, the MUTE_OUT signal is asserted. The MUTE_OUT signal remains asserted until the digital servo loop's internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the MUTE_OUT signal is de-asserted. While MUTE_OUT is asserted, the MUTE_IN signal should be asserted as well to prevent any major distortion in the audio output samples.

Data Format

The serial data input port mode is set by the logic levels on the SRCx_SMODEIN0-2 bits that are located in the SRCCTLx registers. The serial data input port modes available are left-justified, I^2S, TDM and right-justified, 16, 18, 20, or 24 bits.

The serial data output port mode is set by the logic levels on the SRCx_SMODE_OUT0-1 bits. The serial mode can be changed to left-justified, I^2S, right-justified, or TDM. The output word width can be set by using the SRCx_LENOUT0-1 bits. When the output wordwidth is less than 24 bits, dither is added to the truncated bits. The right-justified serial data out mode assumes 64 SCLK cycles per frame, divided evenly for left and right. For the other modes these LSB 8-bits contain zeros. The SRC also supports 16-bit, 32-clock packed input and output serial data in left-justified and I^2S format.

Word Width

The output word width can be set by using the SRCx_LENOUT0-1 bits. When the output word width is less than 24 bits, dither is added to the truncated bits.

Ratio Registesr (SRCRATx)

The SRCRAT0 and SRCRAT1 registers can be read to find the ratio of output to input sampling frequency. This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integerand 11 bits for the fraction. TheSRCRAT register indicates the sample rate ratio of SRCx_FS_OP_I ÷ SRCx_FS_IP_I.

The SRCx_MUTEOUT bits in SRCRATx register report the status of the MUTE_OUT signal. Once the SRCx_MUTEOUT signal is cleared then the ratio can be read.

All SRCx_MUTEOUT bits in SRCRATx register are set after reset.

Operation Modes

The SRC can operate in TDM, I^2S, left-justified, right-justified, matched phase (ADSP-21364 only), and bypass modes. The serial ports of the processor can be used for moving the SRC data to/from the internal memory.

In I^2S, left-justified and right-justified modes, the SRCs operate individually. The serial data provided in the input port is converted to the sample rate of the output port. Figure 12-3 shows the timing in the various formats.

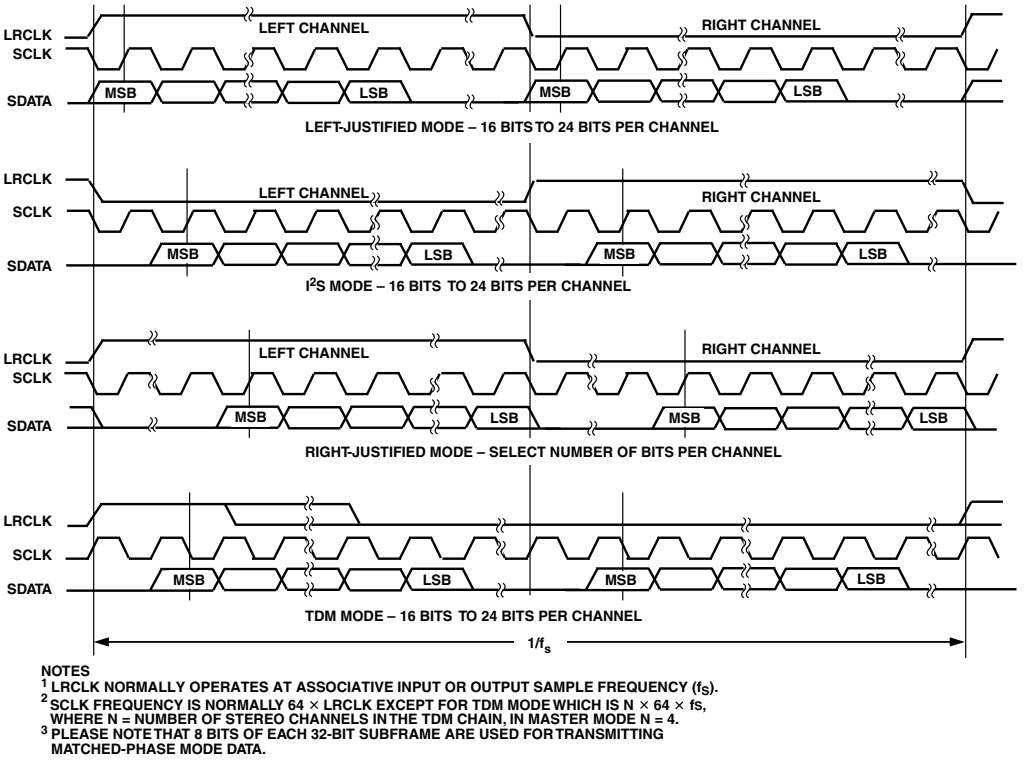

LRCLK
SCLK
SDATA
LEFT CHANNEL
RIGHT CHANNEL
MSB
LSB
LEFT-JUSTIFIED MODE – 16 BITS TO 24 BITS PER CHANNEL
I2S MODE – 16 BITS TO 24 BITS PER CHANNEL
RIGHT-JUSTIFIED MODE – SELECT NUMBER OF BITS PER CHANNEL
TDM MODE – 16 BITS TO 24 BITS PER CHANNEL
1/fs
NOTES
1 LRCLK NORMALLY OPERATES AT ASSOCIATIVE INPUT OR OUTPUT SAMPLE FREQUENCY (fS).
2 SCLK FREQUENCY IS NORMALLY 64 × LRCLK EXCEPT FOR TDM MODE WHICH IS N × 64 × fs, WHERE N = NUMBER OF STEREO CHANNELS IN THE TDM CHAIN, IN MASTER MODE N = 4.
3 PLEASE NOTE THAT 8 BITS OF EACH 32-BIT SUBFRAME ARE USED FOR TRANSMITTING MATCHED-PHASE MODE DATA.

Figure 12-3. SRC Timing

TDM Daisy Chain Mode

The SRCs are daisy chained together to achieve the TDM mode of operation.

TDM Output Daisy Chain

In TDM output port, several SRCs can be daisy-chained together and connected to the SPORT of an ADSP-2136x or other processor (Figure 12-4). The SRC OP contains a 64-bit parallel load shift register. When the `SRCx_FS_OP_I` pulse arrives, each SRC loads its left and right

data into the 64-bit shift register. The input to the shift register is connected to SRCx_TDM_OP_I, and the output is connected to SRCx_DAT_OP_O. By connecting the SRCx_DAT_OP_O to the SRCx_TDM_OP_I of the next SRC, a large shift register is created, which is clocked by SRCx_CLK_OP_I.

TDM Input Daisy Chain

In TDM input port, several SRCs can be daisy-chained together and connected to the serial input port of a SHARC processor or other processor (Figure 12-4). The SRC IP contains a 64-bit parallel load shift register. When the SRCx_FS_IP_I pulse arrives, each SRC parallel loads its left and right data into the 64-bit shift register. The input to the shift register is connected to SRCx_DATA_IP_I, while the output is connected to SRCx_TDM_IP_O. By connecting the SRCx_DATA_IP_I to the SRCx_TDM_IP_O of the next SRC, a large shift register is created, which is clocked by SRCx_IP_CLK_I.

The number of SRCs that can be daisy-chained together is limited by the maximum frequency of SRCx_CLK_xx_I, which is about 25 MHz. For example, if the output sample rate, f_S, is 48 kHz, up to eight SRCs could be connected since 512 f_S is less than 25 MHz.

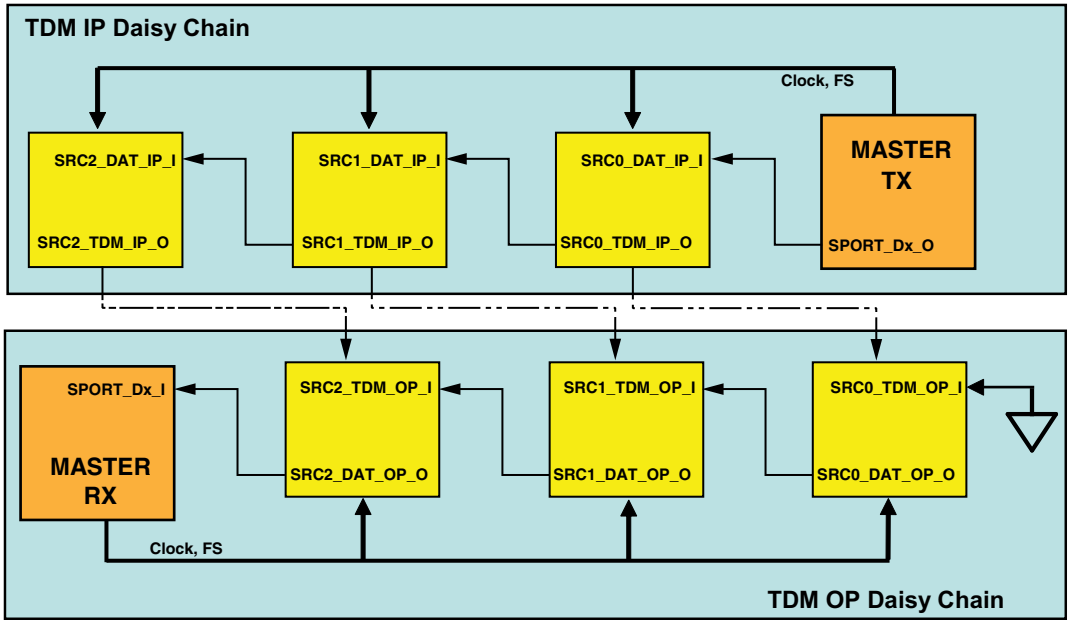


Figure 12-4. TDM Input/Output Modes

Bypass Mode

When the BYPASS bit is set (=1), the input data bypasses the sample rate converter and is sent directly to the serial output port. Dithering of the output data when the word length is set to less than 24 bits is disabled. This mode is ideal when the input and output sample rates are the same and SRCx_FS_IP_I and SRCx_FS_OP_I are synchronous with respect to each other. This mode can also be used for passing through non-audio data since no processing is performed on the input data in this mode.

Matched-Phase Mode (ADSP-21364 Only)

The matched phase mode of the sample rate converter is enabled by the SRCx_MPHASE bit. This mode is used to match the phase (group delay)

between two or more adjacent sample rate converters that are operating with the same input and output clocks. When the SRCx_MPHASE bit is set (=1), the SRC, a matched phase mode slave accepts the sample rate ratio transmitted by another SRC, the matched phase mode master, through its serial output as shown in Figure 12-5.

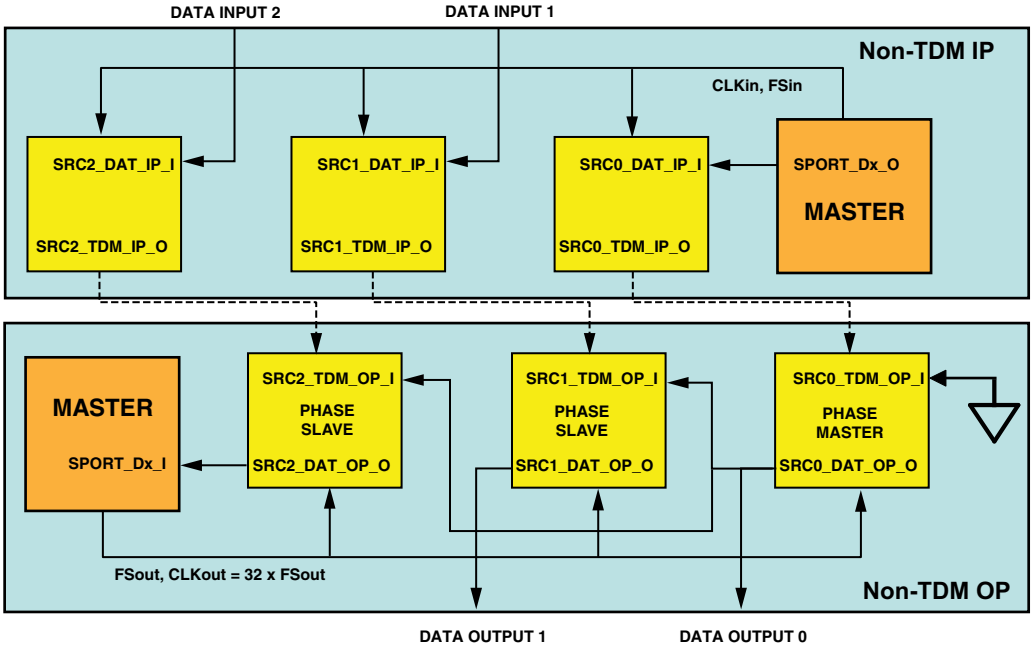


Figure 12-5. Typical Configuration for Matched-Phase Mode Operation

The phase master SRC device transmits its SRCx_FS_OP/SRCx_FS_IP ratio through the data output pin (SRCx_DAT_OP_O) to the slave's SRC's data input pins (SRCx_TDM_OP_I). The transmitted data (32-bit subframe) contains 24-bit data and 8-bits matched phase. The slave SRCs receive the 8-bit matched phase bits (instead of their own internally-derived ratio) if their SRCx_MPHASE bits set to 1, respectively.

The SRCx_FS_IP and SRCx_FS_OP signals may be asynchronous with respect to each other in this mode. Note there must be 32 SRCx_CLK_OP cycles per subframe in matched-phase mode (24-bits data and 8-bits phase match).

Data Format Matched-Phase Mode

The SRC supports the matched-phase mode for all serial output data formats; left-justified, I^{2}S, right-justified, and TDM mode.

Note that in the left-justified, I^{2}S, and TDM modes, the lower 8 bits of each channel subframe are used to transmit the matched-phase data. In right-justified mode, the upper eight bits are used to transmit the matched-phase data. This is shown in Figure 12-6.

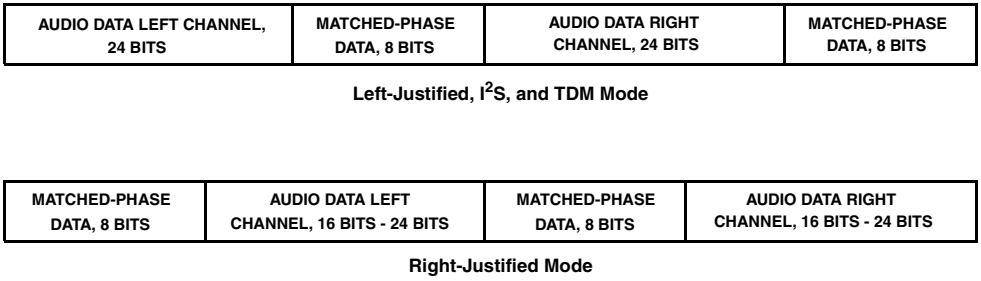

AUDIO DATA LEFT CHANNEL, 24 BITS
MATCHED-PHASE DATA, 8 BITS
AUDIO DATA RIGHT CHANNEL, 24 BITS
MATCHED-PHASE DATA, 8 BITS
Left-Justified, I2S, and TDM Mode
MATCHED-PHASE DATA, 8 BITS
AUDIO DATA LEFT CHANNEL, 16 BITS - 24 BITS
MATCHED-PHASE DATA, 8 BITS
AUDIO DATA RIGHT CHANNEL, 16 BITS - 24 BITS
Right-Justified Mode

Figure 12-6. Matched-Phase Data Transmission

Group Delay

When multiple SRCs are used with the same serial input port clock and the same serial output port clock, the hysteresis causes different group delays (phase mismatches) between multiple SRCs. The filter group delay of the SRC is given by the equations:

$$GDS = \frac{16}{SRCx\_FS\_IP} + \frac{32}{SRCx\_FS\_IP} \text{seconds for } (SRCx\_FS\_OP \geq SRCx\_FS\_IP)$$

$$GDS = \frac{16}{SRCx\_FS\_IP} + \left(\frac{32}{SRCx\_FS\_IP}\right) \times \left(\frac{SRCx\_FS\_IP}{SRCx\_FS\_OP}\right) \text{seconds for } (SRCx\_FS\_OP \leq SRCx\_FS\_IP)$$

Interrupts

The SRC mute-out signal can be used to generate interrupts on their rising edge, falling edge, or both, depending on how the DAI interrupt mask registers (DAI_IRPTL_RE/FE) are programmed. This allows the generation of DAIHI/DAILI interrupts either entering mute, exiting muting or both.

The SRCx_MUTE_OUT interrupt is generated only once when the SRC is locked (after 4096 FS input samples) and after changes to the sample ratio. Hard mute, soft mute, and auto mute only control the muting of the input data to the SRC.

13 PRECISION CLOCK GENERATOR

The precision clock generators (PCG) consist of four units, each of which generates a pair of signals (clock and frame sync) derived from a clock input signal. The units, A and B, are identical in functionality and operate independently of each other. The two signals generated by each unit are normally used as a serial bit clock/frame sync pair.

Features

The following list and Table 13-1 describe the features of the precision clock generators.

Table 13-1. PCG Port Feature Summary

Feature	PCGA–B
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	No
Interrupt Default Routing	N/A
Protocol	
Master Capable	Yes
Slave Capable	No
Transmission Simplex	N/A
Transmission Half Duplex	N/A

Table 13-1. PCG Port Feature Summary (Cont'd)

Feature	PCGA–B
Transmission Full Duplex	N/A
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Interrupt Source	N/A
Boot Capable	N/A
Local Memory	N/A
Clock Operation	CLKIN

Functional Description

The unit that generates the bit clock is relatively simple, since digital clock signals are usually regular and symmetrical. The unit that generates the frame sync output, however, is designed to be extremely flexible and capable of generating a wide variety of framing signals needed by many types of peripherals that can be connected to the signal routing unit (SRU). For more information, see "DAI Group Routing" on page 5-14.

The core phase-locked loop (PLL) has been designed to provide clocking for the processor core. Although the performance specifications of this PLL are appropriate for the core, they have not been optimized or specified for precision data converters where jitter directly translates into time quantization errors and distortion.

As shown in Figure 13-1, the PCG can accept its clock input either directly from the external oscillator (or discrete crystal) connected to the

CLKIN pin or from any of the PCG_EXTx_I (DAI pins). This allows a design to contain an external clock with performance specifications appropriate for the application target.

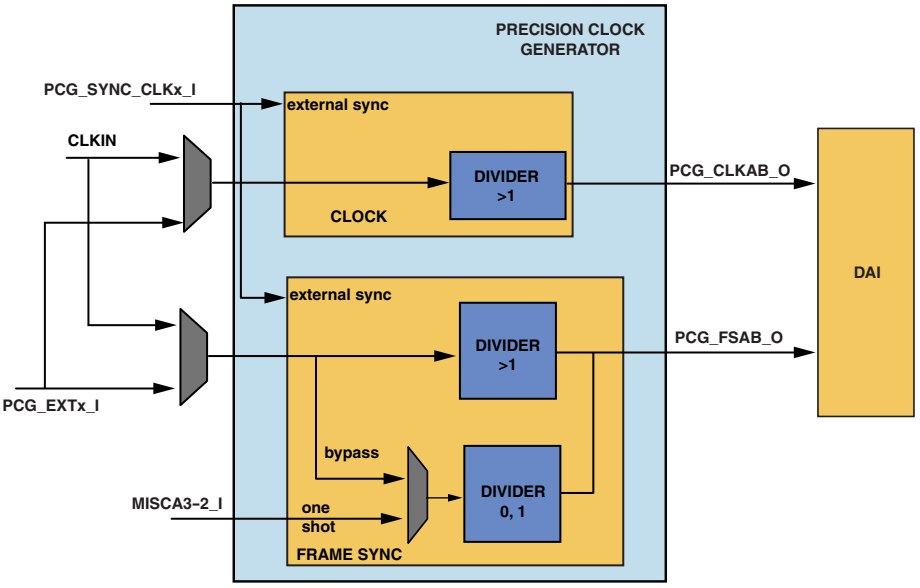


Figure 13-1. PCG Block Diagram

Note that clock and frame sync signals generated by the serial ports are also subject to these jitter problems because the SPORT clock is generated from the core clock. However, a SPORT can produce data output while being a clock and frame sync slave. The clock generated by the SPORT is sufficient for most of the serial communications, but it is suboptimal for analog/digital conversion. Therefore, all precision data converters should be synchronized to a clock generated by the PCG or to a clean (low jitter) clock that is fed into the SRU off-chip via a pin.

Any clock or frame sync unit shouldbe disabled (have its enable bit cleared) before changing any of the associated parameters. After disabling the PCG, a delay of N core clock cycles where N = PCG source clock period ÷ CLKIN period) should be provided before programming PCG with new parameters.

Each of the two units (A and B) produces a synchronization signal for framing serial data. The frame sync outputs are very flexible which allows them to accommodate the wide variety of serial protocols used by the peripherals in the processor.

Pin Descriptions

Table 13-2 provides the pin descriptions for the PCGs. Note x = unit A/B.

Table 13-2. PCG Pin Descriptions

Internal Nodes	I/O	Description
Inputs		
CLKIN	I	External clock input for PCG x
PCG_SYNC_CLKx_I	I	External input used for Frame Synchronization of unit x
PCG_EXTx_I	I	External clock A input provided to the PCG x (not CLKIN)
MISCA2_I	I	External frame sync used for bypass mode PCG A
MISCA3_I	I	External frame sync used for bypass mode PCG B
Outputs		
PCG_CLKx_O	O	Serial clock x output
PCG_FSx_O	O	Frame sync x output

SRU Programming

To use the PCG, route the required inputs using the SRU as described in Table 13-3. Also, use the SRU to connect the outputs to the desired DAI pin.

Table 13-3. PCG DAI/SRU Connections

Internal Nodes	DAI Group	SRU Register
Inputs		
PCG_SYNC_CLKA_I PCG_SYNC_CLKB_I PCG_EXTA_I PCG_EXTB_I	Group A	SRU_CLK4
MISCA2_I MISCA3_I	Group E	SRU_EXT_MISCA
Outputs		
PCG_CLKA_O PCG_CLKB_O	Group A, D, E	
PCG_FSA_O PCG_FSB_O	Group C, D, E	

A PCG clock output cannot be fed to its own input. Setting `SRU_CLK4[4:0] = 28` connects `PCG_EXTA_I` to logic low, not to `PCG_CLKA_O`. Setting `SRU_CLK4[9:5] = 29` connects `PCG_EXTB_I` to logic low, not to `PCG_CLKB_O`.

Register Descriptions

The processor contains registers that are used to control the PCGs.

- **PCG_CTLx0 register.** Enables the clock and frame sync; includes the frame sync divider and the frame sync phase high pulse.

- **PCG_CTLx1 register.** Enables the clock and frame sources, it includes the clock sync divider and the frame sync phase low pulse.
- **PCG_PWx register.** Enables the different operation modes like normal or bypass (either direct bypass or a one shot).
- **PCG_SYNCx register.** Enables the different sources for external synchronization of clock and frame sync.

Clock Inputs

The `CLKxSOURCE` bit (bit 31 in the `PCG_CTLx1` registers) specifies the input source for the clock of the respective units (A, and B). When this bit is cleared (= 0), the input is sourced from the external oscillator/crystal, as shown in Figure 13-1. When set (= 1), the input is sourced from DAI.

Clock Outputs

Each of the four units (A, and B) produces a clock output and a frame sync output. The clock output is derived from the input to the PCG with a 20-bit divisor as shown inthe following equation. Ifthe divisor is zero or one, the PCG's clock generation unit is bypassed, and the clock input is connected directly to the clock output. Otherwise, the PCG unit clock output frequency is equal to the input clock frequency, divided by a 20-bit integer.

$$\textit{Frequency of Clock Output} = \frac{\textit{Frequency of Clock Input}}{\textit{Clock Divisor}}$$

Note that the clock output is always set (as closely as possible) to a 50% duty cycle. If the clock divisor is even, the duty cycle of the clock output is exactly 50%. If the clock divisor is odd, then the duty cycle is slightly less than 50%. The low period of the output clock is one input clock period

more than the high period of the output clock. For higher values of an odd divisor, the duty cycle is close to 50%.

A PCG clock output cannot be fed to its own input.

Frame Sync Outputs

Each of the two units also produces a synchronization signal for framing serial data. The frame sync outputs are much more flexible since they need to accommodate the wide variety of serial protocols used by peripherals.

Frame sync generation from a unit is independently enabled and controlled. Sources for the frame sync generation can be either from the crystal buffer output, or an external pin source. There is only one external source pin for both frame sync and clock output for a unit. If an external source is selected for both frame sync and clock output for a unit, then they operate on the same input signal. Apart from enable and source select control bits, frame sync generation is controlled by a 20-bit divisor, a 16-bit pulse width control, and a 20-bit phase control.

There are two modes of operation for the PCG frame sync. The divisor field determines if the frame sync operates in normal mode (divisor > 1) or bypass mode (divisor = 0 or 1).

Normal Mode

In normal mode, the frequency of the frame sync output is determined by the divisor where:

$$Frequency\ of\ Frame\ Sync\ Output = \left(\frac{Frequency\ of\ Clock\ Input}{Frame\ Sync\ Divisor}\right)$$

The high period of the frame sync output is controlled by the value of the pulse width control. The value of the pulse width control should be less than the value of the divisor.

The phase of the frame sync output is determined by the value of the phase control. If the phase is zero, then the positive edges of the clock and frame sync coincide, provided the divisors of the clock and frame sync are the same, the source for the clock and frame sync is also the same, and if clock and frame sync are enabled at the same time using a single instruction.

The number of input clock cycles that have already elapsed before the frame sync is enabled is equal to the differencebetween the divisor and the phase values. If the phase is a small fraction of the divisor, then the frame sync appears to lead the clock. If the phase is only slightly less than the divisor, then the framesync appears to lag the clock. The frame sync phase should not be greater than the divisor.

Bypass Mode

In bypass mode, the frame sync divisoris either 0 or 1. There are two ways the bypass mode operates, depending on the STROBEA, STROBEB, bits of the PCG_PW register ("Pulse Width Register (PCG_PW)" on page A-67). This is shown below.

- **Direct bypass.** If the STROBEA/B of the PCG_PW register is reset to 0, then the input is directly passed to the frame sync output, either not inverted or inverted, depending on the INVFSA and INVFSB bits of the PCG_PW register.
- **One-shot.** In the bypass mode, if the least significant bit (LSB) of the PCG_PW register is set to 1, then a one-shot pulse is generated. This one-shot-pulse has a duration equal to the period of MISCA2_I for unit A, MISCA3_I for unit B. (For more information, see "Routing Capabilities" on page 5-20.) This pulse is generated either at the rising or at the falling edge of the input clock, depending on the value of the INVFSA and INVFSB bits of the PCG_PW register.

External Trigger Mode

The frame sync output may be synchronized with an external signal by programming the PCG_SYNC register (shown in Figure A-34 on page A-69) and the PCG control registers (PCG_CTLA0-1 and PCG_CTLB0-1) appropriately. In this mode, the rising edge of the external signal starts the frame sync output generation (shown in Figure 13-2).

The trigger with the external clock is enabled by setting bits 0 and 16 of the PCG_SYNC register. The phase must be programmed to 3, so that the rising edge of the external clock is in sync with the frame sync.

Programming should occur in the following order.

1. Program the PCG_SYNC and the PCG_CTLA0-1, PCG_CTLB0-1 registers appropriately.
2. Enable clock or frame sync, or both.

Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed.

All subsequent rising edges will be ignored for subsequent synchronization events.

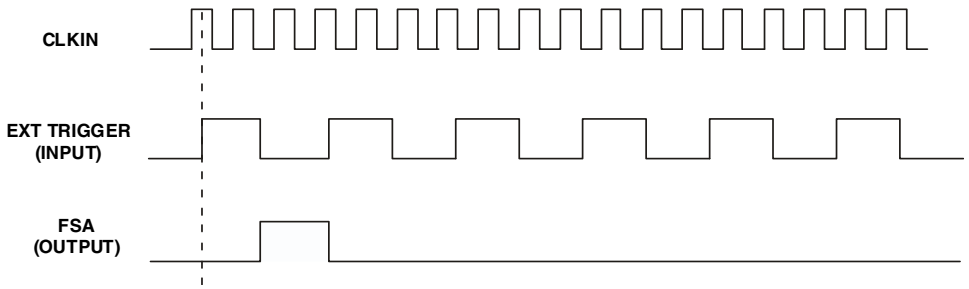


Figure 13-2. FS Output Synchronization With External Trigger

The clock output cannot be aligned with the rising edge of the external clock as there is no phase programmability. Once the clock units have been enabled (by programming bit 1 and bit 17 of the PCG_SYNC register) these outputs are activated when a low-to-high transition is sensed in the external clock (PCG_SYNC_CLKx_I).

Frame Sync

For a given frame sync, the output is determined by the following:

- **Divisor.** A 20-bit divisor of the input clock that determines the period of the frame sync. When set to 0 or 1, the frame sync operates in bypass mode, otherwise it operates in normal mode.
- **Phase.** A 20-bit value that determines the phase relationship between the clock output and the frame sync output. Settings for phase can be anywhere between 0 to DIV – 1.
- **Pulse width.** A 16-bit value that determines the width of the framing pulse. Settings for pulse width can be 0 to DIV – 1. If the pulse width is equal to 0 or the frame sync is even, then the actual pulse width of the output frame sync is:

$$\text{Pulse Width} = \text{Frame Sync Divisor} \div 2$$

For odd divisors the actual pulse width of the output frame sync is:

$$\text{Pulse Width} = \text{Frame Sync Divisor} - 1 \div 2$$

The frequency of the frame sync output is determined by:

$$\text{Frequency of clock input} \div \text{Frame sync divisor}$$

When the divisor is set to any value *other* than 0 or 1, the processors operate in normal mode.

The frame sync divisors (FSxDIV bits) are specified in bits 19–0 of the corresponding PCG control registers (PCG_CTLx0). The pulse width of the frame sync output is equal to the number of input clock periods specified in the 16-bit field of the PCG pulse width register (PCG_PW).

Phase Shift

Phase shift is a frame sync parameter that defines the phase shift of the frame sync with respect to the clock of the same unit. This feature allows shifting of the frame sync signal in time relative to clock signals. Frame sync phase shifting is often required by peripherals that need a frame sync signal to lead or lag a clock signal.

The amount of phase shifting is specified as a 20-bit value in the FSAPHASE_HI bit field (bits 29–20) of the appropriate PCG_CTLA0 register and in the FSAPHASE_LO bit field (bits 29–20) of the PCG_CTLA1 register for unit A. A single 20-bit value spans these two bit fields. The upper half of the word (bits 19–10) resides in the PCG_CTLA0 register, and the lower half (bits 9–0) resides in the PCG_CTLA1 register.

Similarly, the phase shift for frame syncs B is specified in the corresponding PCG_CTLB0 and PCG_CTLB1 registers.

When using a clock and frame syncas a synchronous pair, the units must be enabled in a single atomic instruction before their parameters are modified. Both units must also be disabled in a single atomic instruction.

Phase Shift Settings

The phase shift between clock and frame sync outputs may be programmed under these conditions:

- The input clock source for the clock generator output and the frame sync generator output is the same.

- Clock and frame sync are enabled at the same time using a single atomic instruction.
- Frame sync divisor is an integral multiple of the clock divisor.

If the phase shift is 0, the clock and frame sync outputs rise at the same time. If the phase shift is 1, the frame sync output transitions one input clock period ahead of the clock transition. If the phase shift is divisor – 1, the frame sync transitions divisor – 1 input clock periods ahead of the clock transitions. This translates to the one input clock period after the clock transition (Figure 13-3).

Phase shifting is represented as a full 20-bit value so that even when frame sync is divided by the maximum amount, the phase can be shifted to the full range, from zero to one input clock short of the period.

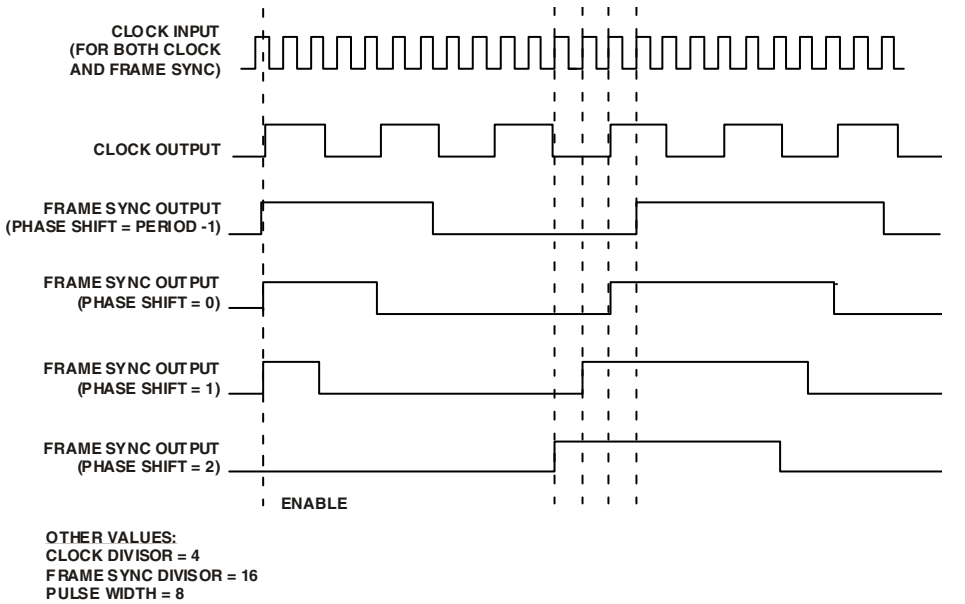


Figure 13-3. Phase Shift Settings

Pulse Width

Pulse width is the number of input clock periods for which the frame sync output is high. Pulse width should be less than the divisor of the frame sync. The pulse width of frame sync A is specified in the PWFSA bits (15–0) of the PCG_PW register and the pulse width of frame sync B is specified in the PWFSB bits (31–16) of the PCG_PW register.

If the pulse width is equal to 0 or if the divisor is even, then the actual pulse width of the frame sync output is equal to:

$$Pulse\ Width = \frac{FrameSyncDivisor}{2}$$

If the pulse width is equal to 0 or if the divisor is odd, then the actual pulse width of the frame sync output is equal to:

$$Pulse\ Width = \frac{FrameSyncDivisor - 1}{2}$$

Bypass Mode

When the divisor for the frame sync hasa value of 0 or 1, the frame sync is in bypass mode, and the PCG_PW register has different functionality than in normal mode. Two bit fields determine the operation in this mode. The one-shot (which is a strobe pulse) frame sync A or B (STROBEx) bit (bits 0 and 16 in the PCG_PW register) determines if the frame sync has the same width as the input, or of a single strobe. The active low frame sync select for the frame syncs (INVFSx) bit (bits 1 and 17 of the PCG_PW register) determines the nature of the output in the simple bypass and single strobe modes as described below. For additional information about the PCG_PW registers, see Figure A-32 on page A-67.

In bypass mode, bits 15–2 and bits 31–18 of the PCG_PWx registers are ignored.

Bypass as a Pass Through

When the STROBEx bit in the PCG_PWx register equals 0, the unit is bypassed and the output equals the input. If, for example, INVFSA (bit 1) for unit A or INVFSB (bit 17) for unit B is set, then the signal is inverted (see Figure 13-4).

Bypass mode also enables the generation of a strobe pulse (one-shot). Strobe usage ignores the counter and looks to SRU to provide the input signal.

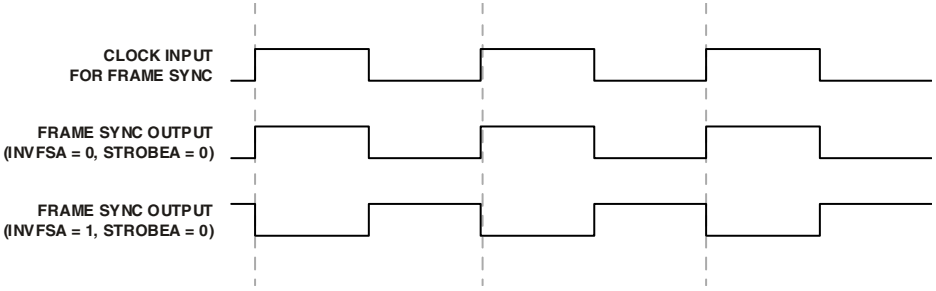


Figure 13-4. Frame Sync Bypass

Bypass as a One-Shot

When the STROBEA or STROBEB bits (bit 0, bit 16 of the PCG_PW register) are set (= 1), the one-shot option is used. When the STROBEx bit is set (= 1), the frame sync is a pulse with a duration equal to one period, or one full cycle of MISCA2_I for unit A or MISCA3_I for unit B, that repeats at the beginning of every clock input period. This pulse is generated during the high period of the input clock when the INVFSA/B bits (bits 1 or 17, respectively of the PCG_PW register) are cleared (INVFSA/B=0) or during the low period of the input clock when invert bits INVFSA/B are set (= 1).

A *strobe period* is equal to the period of the normal clock input signal specified by FSASOURCE (bit 30 in the PCG_CTLA1 register for unit A) and the corresponding FSxSOURCE bit (bit 30 in the PCG_CTLx1 registers for unit B).

As shown in Figure 13-5, the output pulse width is equal to the period of the SRU source signal (MISCA2_I for frame sync A, MISCA3_I for frame sync B). The pulse begins at the second rising edge of MISCAx_I following a rising edge of the clock input. When the INVFSA/B bit is set, the pulse begins at the second rising edge of MISCAx_I, coincident or following a falling edge of the clock input.For more information, see "DAI Group Routing" on page 5-14.

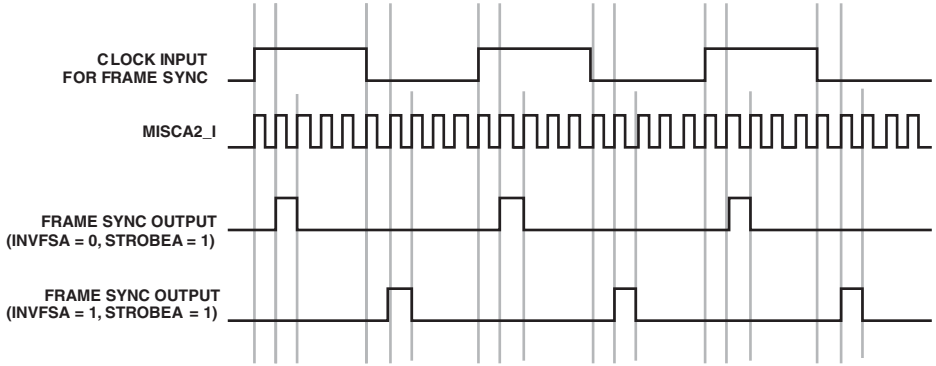


Figure 13-5. One-Shot (Synchronous Clock Input and MISCA2_I)

The second INVFSA bit (bit 1) of the pulse width control register (PCG_PW) determines whether the falling or rising edge is used. When set (= 1), this bit selects an active low frame sync, and the pulse is generated during the low period of clock input. When cleared (=0), this bit is set to active high frame sync and the pulse is generated during the high period of clock input. For more information on the PCG_PWx registers, refer to Table A-30 on page A-67.

Programming Examples

This section contains three programming examples:

1. "Setup for I²S or Left-Justified DAI" on page 13-16
2. "Channel B Clock and Frame Sync Divisors" on page 13-21

Setup for I²S or Left-Justified DAI

This example shows how to set up two precision clock generators using the S/PDIF receiver and an asynchronous sample rate converter (SRC) to interface to an external audio DAC. In this example an input clock (CLKIN) of 33.330 MHz is assumed and the PCG is configured to provide a fixed SRC/DAC output sample rate of 65.098 kHz. The input to the S/PDIF receiver is typically 44.1 kHz if supplied by a CD player but can also be from another source at any nominal sample rate from about 22 kHz to 192 kHz.

Three synchronous clocks are required: a framesync (FSYNC; FS), a master clock (PCGx_CLK; 256 × FS), and a serial bit clock (SCLK; 64 × FS). Since each PCG has only two outputs, this example requires two PCGs. Furthermore, because the digital audio interface requires a fixed-phase relation between SCLK and FSYNC, these two outputs should come from one PCG while the master clock comes from the other.

The CLKIN = 33.330 MHz is divided by the two PCGs to provide the three synchronous clocks — PCGx_CLK, SCLK, and FSYNC for the SRCs and external DAC. These divisors are stored in 20-bit fields in the PCG_CTL registers. For more information, see "Precision Clock Generator Registers" on page A-65.

The integer divisors for several possible sample rates based on 33.330 MHz CLKIN are shown in Table 13-4.

Table 13-4. Precision Clock Generator Division Ratios (33.330 CLKIN)

Sample Rate kHz)	PCG Divisors		
	PCG CLOCK INPUT	SCLK	FSYNC[1]
130.195	1	4	256
65.098	**2**	**8**	**512**
43.398	3	12	768
32.549	4	16	1024
26.039	5	20	1280
21.699	6	24	1536
18.599	7	28	1792

1 The frame sync divisor should be an even integer in order to produce a 50% duty cycle waveform.

For more information, see “Power Management Control Register (PMCTL)” on page A-6. The equation and procedure for programming a master clock input is:

3. Set the core clock rate by meeting the VCO maximum and using the values below (INDIV = 0).

   CCLK = (PLLM ÷ PLLD) × CLKIN
   where:
   PLLM = 12, PLLD = 2 for a CCLK of 200 MHz.
   VCO = 2 × PLLM × CLKIN = 400 MHz.

   Fixed Settings:
   PCLK (peripheral clock) = CCLK ÷ 2 = 100 MHz.
   MCLK (master clock SRC) = PCLK ÷ 4 = 25 MHz.

The combined PCGs can provide a selection of synchronous clock frequencies to support alternate sample rates for the SRCs and external DACs. However, the range of choices is limited by CLKIN and the ratio of PCG_CLKx_O:SCLK:FSYNC which is normally fixed at 256:64:1 to support digital audio, left-justified, I^2S, and right-justified interface modes. Many DACs also support 384, 512, and 786 × FSYNC for PCG_CLKx_O, which allows some additional flexibility in choosing CLKIN.

Note also that in all three DAI modes, the falling edge of SCLK must always be synchronous with both edges of FSYNC. This requires that the phase of the SCLK and FSYNC for a common PCG be adjustable.

While the frequency of PCG_CLKx_O must be synchronous with the sample rate supplied to the external DAC, there is no fixed-phase requirement. For complete timing information, see the *ADSP-2136x SHARC Processor Data Sheet*.

Figure 13-6 shows an example of the internal interconnections between the S/PDIF receiver, SRC, and the PCGs. The interconnections are made by programming the signal routing unit. Note that in this example, CCLK is set at 242 MHz. This frequency can be adjusted up to the maximum CCLK for the chosen processor. Also note that master clock (MCLK) is the input source provided for the PCG. This input can come from CLKIN, any peripheral output, or from one of the DAI pins.

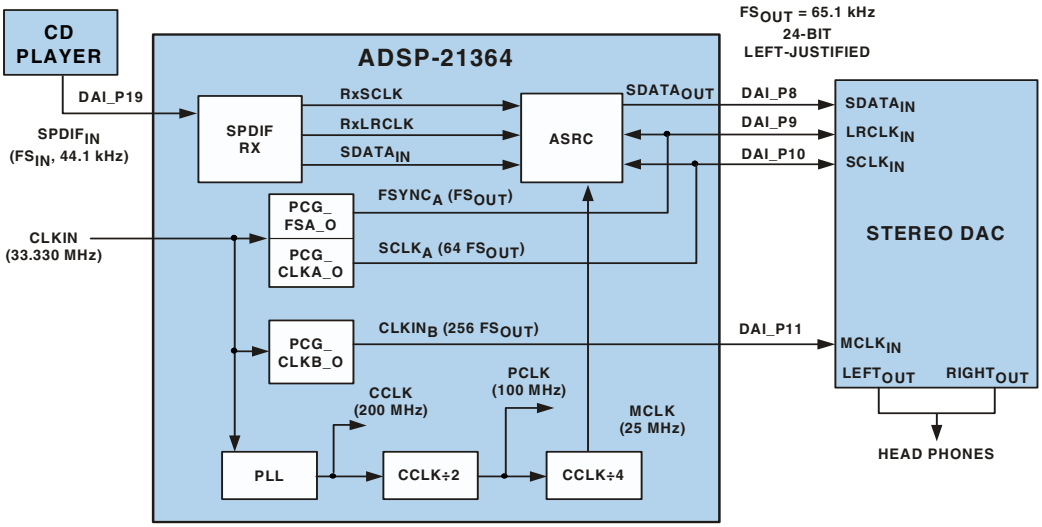


Figure 13-6. PCG Setup for I^{2}S or Left-Justified DAI

In the following example code, the most significant two bits of the control registers (PCG_CTLx) specify the clock source and enable the clock generators. Set the clock divisor and source and low phase word first, followed by the control register enable bits, which must be set together. When the PCG_PW register is set to 0 (default) the FS pulse width is divisor/2 for even divisors and (divisor – 1)/2 for odd divisors. Alternatively, the PCG_PW register could be set high for exactly half the period of CLKIN cycles for a 50% duty cycle, provided the FSYNC divisor is an even number.

Listing 13-1. PCG Initialization

```
/*************************************************************
Required Output Sample Rate = 65.098 kHz
Function              Control    Reg       Phase/   Reg Hex
                      reg        Address   Divisor  Contents
FS_A_Ph_Hi/FS_A_Div   PCG_CTLA0  0x24C0    0/512  0xC00/00200
FS_A_Ph_Lo/CLK_A_Div  PCG_CTLA1  0x24C1    4/8    0x004/00008
```

```
------------------------------------------------------------
FS_B_Ph_Hi/FS_B_Div   PCG_CTLB0  0x24C2  -/-    0x800/00000
FS_B_Ph_Lo/CLK_B_Div   PCG_CTLB1  0x24C3    0/2    0x000/00002
PW_FS_B/PW_FS_A      PCG_PW     0x24C4   0/0    0x0000:0000
************************************************************/
#include  <def21365.h>
/* PCGA --> SCLK & FSYNC Divisors, Sample Rate = 65.098 kHz */
#define PCGA_CLK_DIVISOR  0x0008   /* SCLK output = 64xFs */
#define PCGA_FS_DIVISOR   0x0200   /* FSYNC output = Fs */
#define ENCLKA            0x80000000
#define ENFSA           0x40000000
#define PCGA_FS_PHASE_LO  0x04     /* Set FSYNC/SCLK Phase for
                                      digital audio IF mode */
PCGB --> PCG_CLKx_O Divisor
#define PCGB_CLK_DIVISOR 0x0002    /* PCG_CLKx_O output =
                                      256xFs */
#define PCGB_FS_DIVISOR 0x0000     /* Not used - disabled */
#define PCGB_FS_PHASE_LO 0x00        /* Don't care */
.section/pm seg_pmco; .global Init_PCG;
/***********************************************************/
Init_PCG:
/* Set PCGA SCLK & FSYNC Source first to Xtal Buffer and set
   SCLK_A divisor */
r0 = ((PCGA_FS_PHASE_LO << 20) | PCGA_CLK_DIVISOR);
dm(PCG_CTLA1) = r0;
/* Enable PCGA SCLK & FSYNC and set FSYNC_A divisor */
r0 = (ENCLKA | ENFSA | PCGA_FS_DIVISOR);
dm(PCG_CTLA0) = r0;

/* Set PCGB SCLK & FSYNC Source first to Xtal Buffer and set
   SCLK_B divisor */
r0 = ((PCGB_FS_PHASE_LO << 20) | PCGB_CLK_DIVISOR);
dm(PCG_CTLB1) = r0;

/* Enable PCGB SCLK and disable FSYNC_B */
r0 = (ENCLKB | ENFSB | PCGB_FS_DIVISOR);
dm(PCG_CTLB0) = r0;
```

```
ustat1 = dm(PCG_CTLB0);
bit clr ustat1 ENFSA;
dm(PCG_CTLB0) = ustat1;

/* Set FSYNC_A and FSYNC_B Pulse Width to 50% Duty Cycle
   (default) */
r0 = 0x00000000;
dm(PCG_PW) = r0;
dm(PCG_SYNC) = r0;
Init_PCG.end:
rts;
```

Channel B Clock and Frame Sync Divisors

This section provides two programming examples written for the ADSP-2136x processor. The first listing, Listing 13-2, uses PCG channel B to output a clock on DAI pin 1 and frame sync on DAI pin 2. The input used to generate the clock and frame sync is `CLKIN`. This example demonstrates the clock and frame sync divisors, as well as the pulse width and phase-shift capabilities of the PCG.

Listing 13-2. PCG Channel B Output Example

```
/* Register definitions */
#define SRU_CLK3    0x2434
#define SRU_PIN0    0x2460
#define SRU_PBEN0   0x2478
#define PCG_CTLB0   0x24C2
#define PCG_CTLB1   0x24C3
#define PCG_PW      0x24C4

/* SRU definitions */
#define PCG_CLKB_P      0x39
#define PCG_FSB_P       0x3B
```

```
#define PBEN_HIGH_Of    0x01

//Bit Positions
#define DAI_PBO2     7
#define DAI_PBOE2    6
#define PCG_PWB     16

/* Bit definitions */
#define ENFSB       0x40000000
#define ENCLKB      0x80000000

/* Main code section */
.global _main;
.section/pm seg_pmco;
_main:
/* Route PCG Channel B clock to DAI Pin 1 via SRU */
/* Route PCG Channel B frame sync to DAI Pin 2 via SRU */
r0 = PCG_CLKB_P|(PCG_FSB_P<<DAI_PBO2);
dm(SRU_PIN0) = r0;

/* Enable DAI Pins 1 & 2 as outputs */
r0 = PBEN_HIGH_Of|(PBEN_HIGH_Of<<DAI_PBOE2);
dm(SRU_PBEN0) = r0;

r0 = (100<<PCG_PWB);      /* PCG Channel B FS Pulse width = 100 */
dm(PCG_PW) = r0;

r2 = 1000;                  /* Define 20-bit Phase Shift */
r0 = (ENFSB|ENCLKB|    /*Enable PCG Channel B Clock and FS*/
        1000000);      /* FS Divisor = 1000000 */
r1 = lshift r2 by -10;
/* Deposit the upper 10-bits of the Phase Shift in the */
/* correct position in PCG_CTLB1 (Bits 20-29) */
```

```
r1 = fdep r1 by 20:10;
r0 = r0 or r1;         /* Phase Shift 10-19 = 0 */
dm(PCG_CTLB0) = r0;


r0 = (100000);        /* Clk Divisor = 100000 */
                      /* Use CLKIN as clock source */
/* Deposit the lower 10-bits of the Phase Shift in the */
/* correct position in PCG_CTLB0 (Bits 20-29) */

r1 = fdep r2 by 20:10;
r0 = r0 or r1;         /* Phase Shift 10-19 = 0x3E8 */
dm(PCG_CTLB1) = r0;
//---------------------------------------
_main.end: jump(pc,0);
```

Channel A and B Output Example

Listing 13-3 uses both PCG channels (as opposed to a single channel). Channel A is set up to only generate a clock signal. This clock signal is used as the input to channel B via the SRU. The clock and frame sync are routed to DAI pins 1 and 2, respectively, in the same manner as the first example. The frame sync generated in this example is set for a 50% duty cycle, with no phase shift.

Listing 13-3. PCG Channel A and B Output Example

```
/* Register Definitions */
#define SRU_CLK4    0x2434
#define SRU_PIN0    0x2460
#define SRU_PBEN0   0x2478
#define PCG_CTLA0   0x24C0
#define PCG_CTLA1   0x24C1
```

```
#define PCG_CTLB0   0x24C2
#define PCG_CTLB1   0x24C3
#define PCG_PW      0x24C4

/* SRU Definitions */
#define PCG_CLKA_O      0x1c
#define PCG_CLKB_P      0x39
#define PCG_FSB_P       0x3B
#define PBEN_HIGH_Of    0x01

//Bit Positions
#define PCG_EXTB_I      5
#define DAI_PBO2        6
#define PCG_PWB         16

/* Bit Definitions */
#define ENCLKA      0x80000000
#define ENFSB       0x40000000
#define ENCLKB      0x80000000
#define CLKBSOURCE  0x80000000
#define FSBSOURCE   0x40000000

/* Main code section */
.global _main; /* Make main global to be accessed by ISR */
.section/pm seg_pmco;
_main:
/*Route PCG Channel A clock to PCG Channel B Input via SRU*/
r0 = (PCG_CLKA_O<<PCG_EXTB_I);
dm(SRU_CLK4) = r0;

/* Route PCG Channel B clock to DAI Pin 1 via SRU */
/* Route PCG Channel B frame sync to DAI Pin 2 via SRU */
r0 = (PCG_CLKB_P|(PCG_FSB_P<<DAI_PBO2));
dm(SRU_PIN0) = r0;
```

```
/* Enable DAI Pins 1 & 2 as outputs */
r0 = (PBEN_HIGH_Of|(PBEN_HIGH_Of<<DAI_PBOE2));
dm(SRU_PBEN0) = r0;

r0 = ENCLKA; /* Enable PCG Channel A Clock, No Channel A FS */
           /* FS Divisor = 0, FS Phase 10-19 = 0 */
dm(PCG_CTLA1) = r0;

r1 = 0xfffff; /* Clk Divisor = 0xfffff, FS Phase 0-9 = 0 */
            /* Use CLKIN as clock source */
dm(PCG_CTLA0) = r1;

r0 = (5<<PCG_PWB); /* PCG Channel B FS Pulse width = 1 */
dm(PCG_PW) = r0;

r0 = (ENFSB|ENCLKB|10); /*Enable PCG Channel B Clock and FS*/
            /* FS Divisor = 10, FS Phase 10-19 = 0 */
dm(PCG_CTLB1) = r0;

r0 = (CLKBSOURCE|FSBSOURCE|10); /* Clk Divisor = 10 */
     /* FS Phase 0-9 = 0, Use SRU_MISC4 as clock source */
dm(PCG_CTLB0) = r0;

_main.end: jump(pc,0);
```

14 SYSTEM DESIGN

The ADSP-21362/3/4/5/6 processors support many system design options. The options implemented in a system are influenced by cost, performance, and system requirements. This chapter provides the following system design information:

- "Conditioning Input Signals" on page 14-2
- "Clocking" on page 14-4
- "Power-Up Sequence" on page 14-11
- "Processor Pin Descriptions" on page 14-16
- "System Components" on page 14-26
- "Designing for High Frequency Operation" on page 14-28
- "Processor Booting" on page 14-32

Other chapters also discuss system design issues. Some other locations for system design information include:

- "Pin Descriptions" on page 6-6
- "Pin Descriptions" on page 7-3

By following the guidelines described in this chapter, you can ease the design process for your ADSP-2136x processor product. Development

and testing of your application code and hardware can begin without debugging the JTAG port.

Before proceeding with this chapter it is recommended that you become familiar with the ADSP-2136x processor core architecture. This information is presented in the *SHARC Processor Programming Reference*.

Conditioning Input Signals

The processor is a CMOS device. It has input conditioning circuits which simplify system design by filtering or latching input signals to reduce susceptibility to glitches or reflections. The following sections describe why these circuits are needed and their effect on input signals.

A typical CMOS input consists of an inverter with specific N and P device sizes that cause a switching point of approximately 1.4 volts. This level is selected to be the midpoint of the standard TTL interface specification of V_{IL} = 0.8 V and V_{IH} = 2.0 V. This input inverter,unfortunately, has a fast response to input signals and external glitches wider than 1ns. Filter circuits and hysteresis are added after the input inverter on some processor inputs, as described in the following sections.

Reset Input Hysteresis

Hysteresis is used only on the $\overline{\text{RESET}}$ input signal. Hysteresis causes the switching point of the input inverter to be slightly above 1.4 V for a rising edge and slightly below 1.4V for a falling edge. Thevalue of the hysteresis is approximately ± 0.1 V. The hysteresis is intended to prevent multiple triggering of signals which are allowed to rise slowly, as might be expected on a reset line with a delay implemented by an RC input circuit. Hysteresis is not used to reduce the effect of ringing on processor input signals with fast edges, because the amount of hysteresis that can be used on a CMOS chip is too small to make a difference. The small amount of hys-

teresis allowed is due to the restrictions on the tolerance of the V_{IL} and V_{IH} TTL input levels under worst-case conditions.

Refer to the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet* for exact specifications.

Clock Input Specifications and Jitter

The clock input signal must be free of ringing and jitter. Clock jitter can easily be introduced into a system where more than one clock frequency exists. Jitter should be kept to an absolute minimum. High frequency jitter on the clock to the processor may result in abbreviated internal cycles.

Keep the portions of the system that operate at different frequencies as physically separate as possible. The clock supplied to the processor must have a maximum rise time and must meet or exceed a high and low voltage of V_{IH} and V_{IL}, respectively.

Refer to the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet* for exact specifications.

Input Synchronization Delay

The processor has several asynchronous inputs:

- $\overline{\texttt{RESET}}$
- $\overline{\texttt{TRST}}$
- FLAG3-0
- AD15-0
- DAI_PB[20-1]

These inputs can be asserted in arbitrary phase to the processor clock, CLKIN. The processor synchronizes the inputs prior to recognizing them. The delay associated with recognition is called the synchronization delay.

Any asynchronous input must be valid prior to the recognition point in a particular cycle. If an input does not meet the setup time on a given cycle, it may be recognized in the current cycle or during the next cycle.

To ensure recognition of an asynchronous input, it must be asserted for at least one full processor cycle plus setup and hold time, except for $\overline{\text{RESET}}$, which must be asserted for at least four CLKIN processor cycles. The minimum time prior to recognition (the setup and hold time) is specified in the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

Never share a clock buffer IC with a signal of a different clock frequency. This introduces excessive jitter.

Clocking

To provide the clock generation for the core and system, the processor uses an analog PLL with programmable state machine control. The PLL design serves a wide range of applications. It emphasizes embedded applications and low cost for general-purpose processors, in which performance, flexibility, and control of power dissipation are key features. This broad range of applications requires a range of frequencies for the clock generation circuitry. The input clock may be a crystal, an oscillator, or a buffered, shaped clock derived from an external system clock oscillator.

Figure 14-1 illustrates a conceptual model of the PLL circuitry with configuration inputs and resulting outputs. In the figure, the VCO (voltage controlled oscillator) is an intermediate clock from which the core clock (CCLK) and peripheral clock (PCLK) are derived. Furthermore, an implemented bypass path allows programs to change any VCO frequency dynamically while the core is fed at CLKIN speed.

Subject to the maximum VCO frequency, the PLL supports a wide range of multiplier ratios of the input clock, CLKIN. To achieve this wide multiplication range, the processor uses a combination of programmable multipliers in the PLL feedback circuit and output configuration blocks.

The power management control register (PMCTL) governs the operation of the PLL. For details, see "Power Management Control Register (PMCTL)" on page A-6.

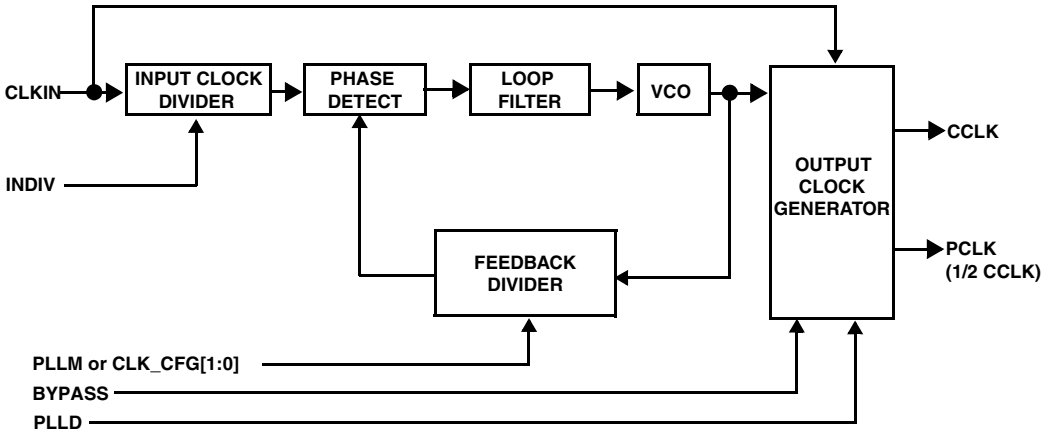


Figure 14-1. PLL Block Diagram

Input Clock

The processor receives its clock input on the CLKIN pin. As a source, CLKIN can be driven from:

- an external crystal oscillator (connected to the CLKIN pin)
- a crystal (connected between the XTAL and CLKIN pins)

The processor uses an on-chip, phase-locked loop (PLL) to generate its internal clock, which is a multiple of the CLKIN frequency. Because the

PLL requires some time to achieve phase lock, CLKIN must be valid for a minimum time period during reset before the $\overline{\text{RESET}}$ signal can be deasserted. For information on minimum clock setup, see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

When using an external crystal, the maximum crystal frequency cannot exceed 25 MHz. The internalclock generator, when used in conjunction with the XTAL pin and an external crystal, is designed to support a maximum 25 MHz external crystal frequency. For all other external clock sources, the maximum CLKIN frequency is 50 MHz.

The *minimum* operational range for any given CLKIN frequency is constrained by the operating range of the PLL. For complete timing information, consult the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

Input Clock Divider

Divides the PLL input clock by 2 if enabled (INDIV bit). Since changing the PLL input clock affects the VCO frequency, the PLL must be put in bypass mode before the change is made.

Feedback Divider

The PLL feedback divider block is controlled by both hardware and software based on the PLL multiplier settings below.

- Hardware—through the clock configuration pins (CLK_CFG[1:0])
- Software—through the PLLM bits

Hardware Control

On power-up, the CLK_CFG[1:0] pins are used to select ratios of 32:1, 16:1, and 6:1 which cannot be changed during runtime. After booting however, numerous other ratios (slowing or speeding up the clock) can be selected through software control.

Table 14-1 describes the internal clock to CLKIN frequency ratios supported by the processor.

Table 14-1. Pin Selectable Clock Rate Ratios

CLKCFG1-0	Core to CLKIN Ratio
00	6:1
01	32:1
10	16:1
11	Reserved

Table 14-2 demonstrates the internal core clock switching frequency across a range of CLKIN frequencies. The minimum operational range for any given frequency may be constrained by the operating range of the phase-locked loop. Note that the goal in selecting a particular clock ratio for an application is to provide the highest permissible internal frequency for a given CLKIN frequency. For more information on available clock rates, see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

Table 14-2. Selecting Core to CLKIN Ratio

	Typical Crystal and Clock Oscillators Inputs					
	12.500	16.667	25.000	33.333	40.000	50.000
Clock Ratios	**Core CLK (MHz)**[1]					
6:1	N/A	100	150	200	240	300
16:1	200	266.66	N/A	N/A	N/A	N/A
32:1	N/A	N/A	N/A	N/A	N/A	N/A

1 For operational limits forthe core clock frequency see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

Software Control

Programs control the PLL through the PMCTL register. The PLL multiplier (PLLM) bits can be configured to set a multiplier range of 0 to 63. This allows the PLL to be programmed dynamically in software to achieve a higher or slower core instruction rate depending on a particular system's requirements.

The reset value of the PLLM bits is derived from the CLK_CFG[1:0] pin multiply ratio settings. This value can be reprogrammed in the boot kernel to take effect immediately after startup.

VCO Clock

The VCO is the PLL output stage of the PLL. It feeds the output clock generator which provides core and peripheral clocks as shown in Table 14-3. Two settings have an impact on the VCO frequency:

- The INDIV bit enables the CLKIN input divide by 2

- The PLLM bits ÷ CLK_CFG[1:0] pins control the PLL feedback divider unit

Table 14-3. VCO Encodings

PLLM Bit Settings	VCO Frequency[1]	
	INDIV = 0	INDIV = 1
0	64x	32x
1	1x	0.5x
2	2x	1x
N = 3–62	Nx	0.5Nx
63	63x	31.5x

1 For operational limits for the VCO clock see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

Output Clock Generator

The output clock generator divides and synchronizes all output clocks. It comes after the VCO and does not provide any feedback to the VCO circuit. This block also ensures the transition from bypass to non bypass mode. For a description of the PLLD bits, see "Power Management Control Register (PMCTL)" on page A-6.

New divisor ratios are picked up on the fly and the clocks smoothly transition to their new values within 14 core clock (CCLK) cycles.

Only VCO frequency changes require bypass mode. This does not apply to the output clock generator unit.

Core Clock (CCLK)

The PLL divider bits (PLLD) divides the PLL output clock to create the processor core clock (CCLK). The divisor can be changed any time and new

division ratios are implemented on the fly without entering bypass mode (since the VCO frequency does not change).

The reset value for the output divisor of the PLLD bit field is 1. This value can be reprogrammed in the boot kernel to take effect immediately after startup.

Peripheral Clock (PCLK)

The peripheral clock is derived from the core clock with a fixed divisor of 2. This clock feeds all the peripherals including the I/O processor (IOP).

Bypass Clock

Bypass mode must be used if any runtime VCO clock change is required. Setting the PLLBP bit bypasses the entire PLL circuitry. In bypass mode, the core runs at CLKIN speed. Once the PLL has settled into the new VCO frequency, (which may take 4096 CLKIN cycles) the PLLBP bit may be cleared to release the core from bypass mode.

Only VCO frequency changes require bypass mode. This does not apply to the output divider block.

Power Savings

The PMCTL register allows programs to disable the clock source to a particular peripheral (for example the SPORTs or the SPI) to further conserve power. By default, each peripheral block has its internal clock enabled only after it is initialized. Programs can use the PMCTL register to turn the specific peripheral when not needed. After reset, these clocks are not enabled until the peripheral itself is initialized by the program.

Power Supplies

The ADSP-2136x has a separate power supply connection for the core (V_{DDINT}) and I/O (V_{DDEXT}). For more information on power consumption, voltage levels and power-up timing, see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

Power Supply for the PLL

The ADSP-2136x on-chip PLL is supplied with two separate pins for analog voltage (A_{VDD}) and analog GND (A_{VSS}). For better noise immunity of the PLL circuit, an external analog filter circuit is recommended. For more information, see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

Power-Up Sequence

The proper power-up sequence is critical to correct processor operation as described in the following sections.

Input Clock

If an external clock oscillator is used, it should NOT drive the `CLKIN` pin when the processor is not powered (similar to a hibernate mode). The clock must be driven immediately after power-up; otherwise, internal gates stay in an undefined (hot) state and can draw excess current. After power-up, there should be sufficient time for the oscillator to start up, reach full amplitude, and deliver a stable `CLKIN` signal to the processor before the reset is released. This may take several milliseconds and depends on the choice of crystal, operating frequency, loop gain and capacitor ratios. For details on timing, refer to the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

PLL Start-Up

Before the PLL can start settling, the $\overline{\text{RESET}}$ signal should be asserted for several micro-seconds under the following conditions. For PLL information, see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

- valid and stable core voltage (VDDINT)
- valid and stable I/O voltage (VDDEXT)
- valid and stable clock input (CLKIN)

The chip reset circuit is shown in Figure 14-2. The PLL needs time to lock to the CLKIN frequency before the core can execute or begin the boot process. A delayed core reset signal ($\overline{\text{RESETOUT}}$) is triggered by a 12-bit counter after $\overline{\text{RESET}}$ is transitioned from low to high (approximately 400 µs for $\text{CLKIN}_{\text{min}}$). The delay circuit is activated at the same time the PLL is triggered for settling after reset is de-asserted.

After the external processor $\overline{\text{RESET}}$ signal is deasserted, the PLL starts settling. The rest of the chip is held in reset for 4096 CLKIN cycles after $\overline{\text{RESET}}$ is deasserted by an internal reset signal.

In addition to the hardware reset, there is also support for a software reset, which can be asserted by setting the SRST bit of the SYSCTL register (see "System Control Register (SYSCTL)" on page A-4). This has the same affect as the hardware reset.

The advantage of the delayed core reset is that the PLL can be reset any number of times without having to power down the system. If there is a brownout situation, the external watchdog circuit only has to control the $\overline{\text{RESET}}$ signal. For more information on device power-up, see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

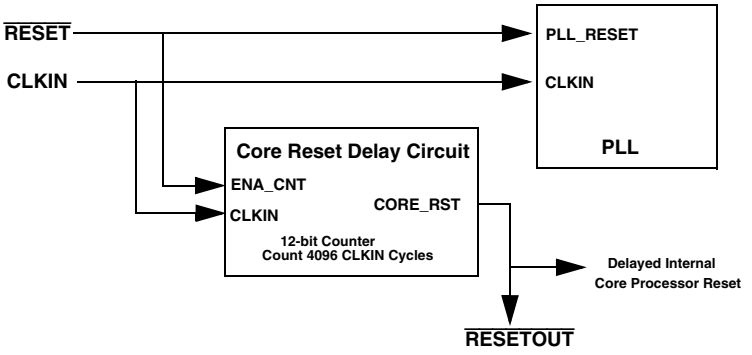


Figure 14-2. Chip Reset Circuit

Examples for Power Management

The following examples show different methods for using the power saving features in the SHARC processors.

Listing 14-1. Power Savings for the SPI Module

```
ustat2 = dm(PMCTL);
bit set ustat2 SPIPDN;   /* disable internal peripheral clock for
                            SPI module. /
dm(PMCTL) = ustat2;
```

Example for Output Divider Management

Listing 14-2. Using the Output Divisor

```
ustat2 = dm(PMCTL);
bit set ustat2 DIVEN|PLLD4;   /* set and enable output Divisor to
                                 4 for CoreCLK = (CLKIN x M/D) =
                                  (CLKIN x M/4) */
dm(PMCTL) = ustat2;
```

Examples For VCO Clock Management

There are two allowable procedures to program the VCO. The first is:

1. Set the PLL multiplier and divisor value and enable the divisor by setting the DIVEN bit.
2. After one core clock cycle, place the PLL in bypass mode by setting (= 1) the PLLBP bit.
3. Wait in bypass mode until the PLL locks.
4. Take the PLL out of bypass mode by clearing (= 0) the bypass bit.

Use the following alternate procedure to program the PLL.

1. Set the PLL multiplier and divisor values and place the PLL in bypass mode by setting the PLLBP bit.
2. Wait in the bypass mode until the PLL locks.
3. Take the PLL out of bypass mode by clearing the bypass bit.
4. Wait for one core clock cycle.
5. Enable the divisor by setting the DIVEN bit.

Listing 14-3. PLL Programming Example 1

```
ustat2 = dm(PMCTL);
bit set ustat2 DIVEN | PLLD4 |PLLM16;  /* set a multiplier of
                                          16 and a divider of 4 */
dm(PMCTL) = ustat2;
bit set ustat2 PLLBP;       /* Put PLL in bypass mode. */
bit clr ustat2 DIVEN;        /* clear the DIVEN bit */
dm(PMCTL) = ustat2;         /* The DIVEN bit should be cleared
                             while placing the PLL in bypass mode */
waiting_loop:
```

```
r0 = 4096;                /* wait for PLL to lock at new rate
                            (requirement for VCO change) */
lcntr = r0, do pllwait until lce;
pllwait: nop;
ustat2 = dm(PMCTL);     /* Reading the PMCTL register value
                           returns the DIVEN bit value as zero */
bit clr ustat2 PLLBP;   /* take PLL out of Bypass, PLL is now at
                CLKIN*4 (CoreCLK = CLKIN * M/D = CLKIN * 16/4) */

dm(PMCTL) = ustat2;    /* The DIVEN bit should be cleared while

                          taking the PLL out of bypass mode */
```

Listing 14-4. PLL Programming Example 2

```
ustat2 = dm(PMCTL);
bit set ustat2 PLLBP | PLLD4 |PLLM16;
/* set a multiplier of 16 and a divider of 4 and enable Bypass
mode*/
waiting_loop:
r0 = 4096;          /* wait for PLL to lock at new rate
                         (requirement for VCO change) */

lcntr = r0, do pllwait until lce;
pllwait: nop;
ustat2 = dm(PMCTL);
bit clr ustat2 PLLBP;      /* take PLL out of Bypass*/
dm(PMCTL) = ustat2;
ustat2 = dm(PMCTL);
bit set ustat2 DIVEN;

/* Enable the DIVEN bit, PLL is now at
   CLKIN*4 (CoreCLK = CLKIN * M/D = CLKIN* 16/4) */
dm(PMCTL) = ustat2;
```

RESET Function

A reset is required to place the processor into a known good state out of power-up. Table 14-4 shows the differences between a hardware reset ($\overline{\texttt{RESET}}$ pin de-asserted) or a software reset (setting bit 0 in the SYSCTL register).

Table 14-4. Reset Function

Function	Hardware Reset	Software Reset
PLL	Yes	No
$\overline{\texttt{RESETOUT}}$ pin asserted	4096 CLKIN Cycles	2 PCLK Cycles
Core	Yes	Yes
IOP	Yes	Yes
Memory	Yes	No
Booting	Yes	Yes
PMCTL Register	Yes	No
Emulator Registers	Yes	No

Processor Pin Descriptions

Refer to the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet* for pin information, including package pinouts for the currently available package options.

JTAG Interface Pins

The JTAG Test Access Port (TAP) consists of the TCK, TMS, TDI, TDO, and TRST pins. The JTAG port can be connected to a controller that performs a boundary scan for testing purposes. This port is also used by the Analog Devices processor tools product line of JTAG emulators and development software to access on-chip emulation features. To allow the use of the

emulator, a connector for its in-circuit probe must be included in the target system.

If the TRST pin is not asserted (or held low) at powerup, the JTAG port is in an undefined state that may cause the SHARC processor to drive out on I/O pins that would normally be three-stated at reset. The TRST pin should be tied low with a pull-down resistor. For more information, see the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

A detailed discussion of JTAG can be found in Engineer-to-Engineer Note EE-68, *Analog Devices JTAG Emulation Technical Reference*. This document is available on the Analog Devices Web site at www.analog.com.

Pin Impedance

Like previous SHARC processors, the ADSP-2136x processors contain internal series resistance equivalent to 360 Ohms on the input path of all pins.

Pin Multiplexing

The ADSP-2136x processors provide the same functionality as other SHARC processors but with a much lower pin count (reducing system cost). It does this through extensive use of pin multiplexing. Table 14-5 shows the registers and the associated bits that are used in multiplexing.

Table 14-5. Multiplexing Registers and Bits

Registers Used	Bits Used
SYSCTL	PPFLGS, TMREXPEN, IRQxEN, FLGxEN, PWMxEN
SPIFLG, SPIFLGB	SPIFLG15–0
SPICTL, SPICTLB	SPIMS

Table 14-5. Multiplexing Registers and Bits (Cont'd)

Registers Used	Bits Used
IDP_PP_CTL	IDP_PP_SELECT, PDAP_SEL
PPCTL	PPEN

FLAG3–0 Pins

As described Table 14-5 and shown in Figure 14-3, The FLAG3-0 pins can multiplex around the following four interfaces.

- FLAGS (input/output)
- Interrupts (input)
- Core timer (output)
- SPI (output, slave selects)

The FLAG3-0 pins allow single bit signaling between the processor and other devices. For example, the processor can raise an output flag to interrupt a host processor. Each flag pin can be programmed to be either an input or output. In addition, many processor instructions can be conditioned on a flag's input value, enabling efficient communication and synchronization between multiple processors or other interfaces.

The flags are bidirectional pins and all have the same functionality. The FLG3-0 bits in the FLAGS register program the direction of each flag pin. For more information, see the *SHARC Processor Programming Reference*.

The processor's external interrupt pins, and core timer pin can be used to send and receive control signals to and from other devices in the system. The IRQ2-0 pins are mapped on the FLAG2-0 pins and the TMREXP pin (core timer) is mapped on the FLAG3 pin. Hardware interrupt signals (IRQ2-0) are received on the FLAG2-0 pins. Interrupts can come from devices that require the processor to perform some task on demand. A memory-mapped peripheral, for example, can use an interrupt to alert the

processor that it has data available. For more information, see Appendix B, Interrupts.

The TMREXP output is generated by the on-chip core timer. It indicates to other devices that the programmed time period has expired. For information on core timer, see the *SHARC Processor Programming Reference*.

Table 14-6. FLAG3–0 Pin Multiplexing Scheme

Control	Function of FLAG3–0	Type/Comment
PPFLGS = 0 IRQxEN = 0 TMREXPEN = 0 DSxEN = 0	GPIO	FLAG3–0 (input/output) act as GPIO pins.
PPFLGS = 0 IRQxEN = 1 TMREXPEN = 1 DSxEN = 0	IRQ2–0/ TMREXP	FLAG2–0 (input) act as IRQ2–0 interrupt. FLAG3 (output) pin act as core timer expire (TMREXP).
PPFLGS = 1 IRQxEN = 1 TMREXPEN = 1 DSxEN = 0	IRQ2–0/ TMREXP	FLAG2–0 (input) act as IRQ2–0 interrupt. FLAG3 (output) pin act as core timer expire (TMREXP). FLAG3-0 GPIO functionality has been moved to the AD11-8 pins.
PPFLGS = 0 IRQxEN = x TMREXPEN = x DSxEN = 1	SPI slave select 3–0	FLAG3–0 (output) pin act as SPI slave select.

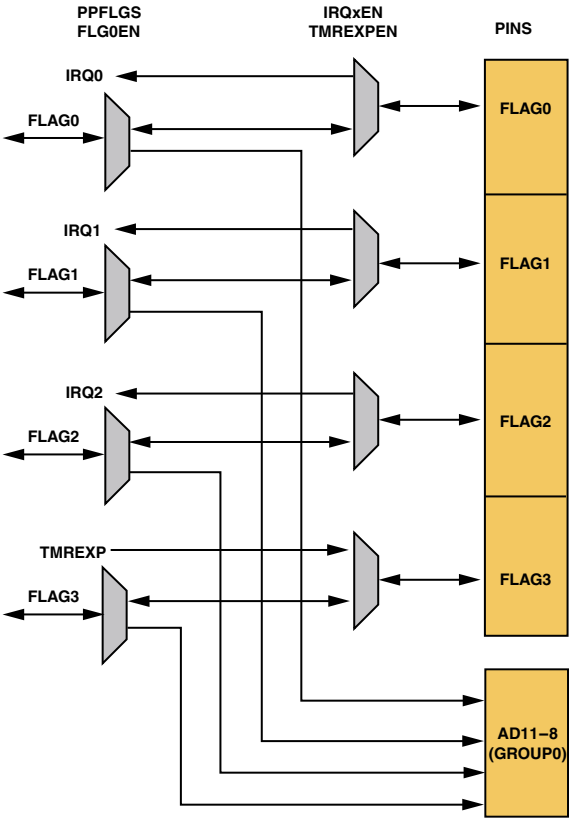


Figure 14-3. FLG3–0/IRQ2–0/TMREXP Pin Multiplexing Scheme

Parallel Port Pin Multiplexing

As described Table 14-7 and shown in Figure 14-4, the parallel port can multiplex around the following five interfaces.

- Parallel port address and data (input/output)
- PDAP (input only)
- FLAGS (input/output)

- PWM channels (output)
- SPI (output, slave selects)

To use the muxed parallel port pins as flags (FLAG15-0), set (= 1) the PPFLGS (bit 20) of the SYSCTL register and disable the parallel port by clearing (=0) the PPEN bit (bit 0) in the PPCTL register. For the address pin to FLAG pin mapping, refer to the *ADSP-21362/3/4/5/6 SHARC Processor Data Sheet*.

In the PDAP control register (IDP_PP_CTL), the IDP_PP_SELECT bit (bit 26) is the logical AND of the IDP_PDAP_EN bit (bit 31). Setting the IDP_PP_SELECT bit (=1) selects the 16 inputsfrom the parallel portAD15-0. clearing this bit (=0) selects the 16 inputs from DAI pin buffers DAI_P20-5.

Table 14-7. Parallel Port Pin Multiplexing Scheme

Control F	unction of AD15–0	Type/Comment
IDP_PDAP_EN = 0 PPEN = 0 PPFLGS = 0 FLAGxEN = 0 PWMxEN = 0	Three-state	Not connected.
IDP_PDAP_EN = 1 PPEN = x PPFLGS = x FLAGxEN = x PWMxEN = x	PDAP	Input. PDAP input has highest priority.
IDP_PDAP_EN = 0 PPEN = 1 PPFLGS = x FLAGxEN = x PWMxEN = x	Parallel Port	Input/Output.

Table 14-7. Parallel Port Pin Multiplexing Scheme (Cont'd)

Control F	unction of AD15–0	Type/Comment
IDP_PDAP_EN = 0 PPEN = 0 PPFLGS = 1 FLAGxEN = 1 PWMxEN = x	FLAGx	Input. FLAG higher priority then PWM. FLAG can move in groups of 4.
IDP_PDAP_EN = 0 PPEN = 0 PPFLGS = 1 FLAGxEN = 0 PWMxEN = 1	PWMx	Input. PDAP input has highest priority. PWM can move in groups of 4.

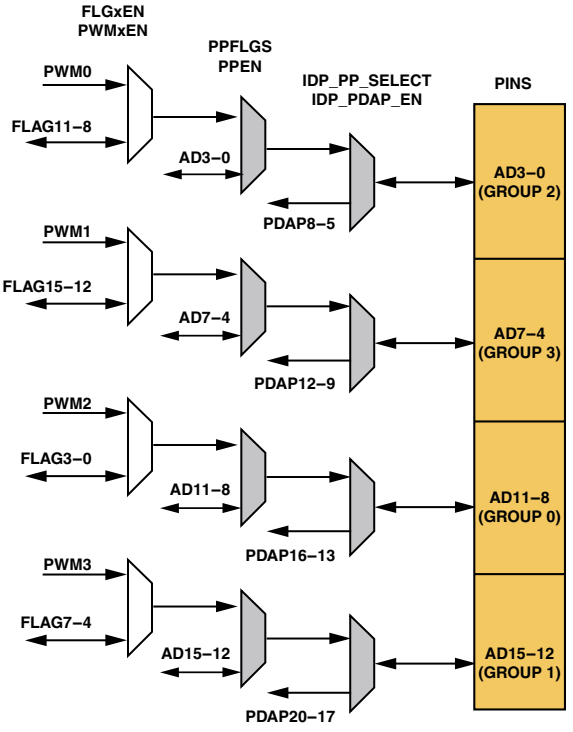


Figure 14-4. Parallel Port Pin Multiplexing Scheme

SPI Master Slave Select

There are two SPI interfaces on-chip—the primary SPI which allows connectivity to the FLAG3-0 pins, the parallel port and the DAI pins. The secondary SPI, (SPIB) only routes slave selects to the DAI pins (Figure 14-5).

The serial peripheral interface can use up to four general-purpose I/O pins FLAGx. SPI functionality is defined in terms of general-purpose I/O, not just pins. Therefore, when the SPI is using FLAGx, there are only 12 general-purpose I/O channels available. When the FLAGx general-purpose I/O is transferred to the parallel port pins, SPI function also moves to the parallel port pins.

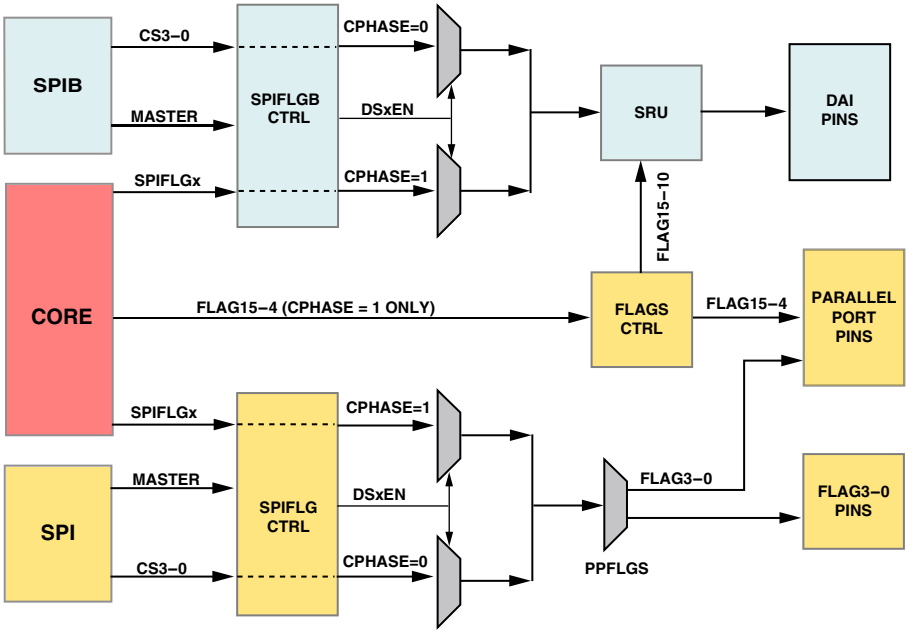


Figure 14-5. SPI Slave Select Pin Multiplexing

Parallel Port and DAI Pin Multiplexing Scheme

Note that the FLAG15-10 direction paths from the core to the parallel port and DAI pins operate in parallel.

- In output mode, if the same flag is mapped to both parallel port pins and DAI pins, then the output is driven from both pins.
- In input mode, if the same flagis mapped to both parallel port pins and DAI pins, then the input from the parallel port pins has priority.

The FLAG15-10 signals (configured in the FLAGS register) are internally connected to DAI_INT_25-31 signals (SRU_EXTMISCA/B routing registers). Signal changes will update the status in both registers.

PWM Multiplexing Scheme

Table 14-8 shows how to connect the PWM outputs on the parallel port pins.

Table 14-8. PWM Connection on the Parallel Port

Control Bits	Pin Multiplexing	PWM Unit
PWM2EN=1	AD0=AL0 AD1=AH0 AD2=BL0 AD3=BH0	PWM0
PWM3EN=1	AD4=AL1 AD5=AH1 AD6=BL1 AD7=BH1	PWM1

Table 14-8. PWM Connection on the Parallel Port (Cont'd)

Control Bits	Pin Multiplexing	PWM Unit
PWM0EN=1	AD8=AL2 AD9=AH2 AD10=BL2 AD11=BH2	PWM2
PWM1EN=1	AD12=AL3 AD13=AH3 AD14=BL3 AD15=BH3	PWM3

SRU Flag Description

Table 14-9 describes how to route the general-purpose I/O flags within the SRU. Note only the `FLAG15-10` can be used for this purpose.

Table 14-9. GPIO/SRU Routing

Internal Node	DAI Connection	SRU Register
Inputs		
FLAG15–13_I	Group E	SRU_EXT_MISCA
FLAG12–10_I		SRU_EXT_MISCB
Outputs		
FLAG15–10_O	Group D	
FLAG15–10_PBEN_O	Group F	

System Components

This section provides some recommendations for other components to use when designing a system for your ADSP-21362/3/4/5/6 processor.

Supervisory Circuits

It is important that a processor (or programmable device) have a reliable active $\overline{\text{RESET}}$ that is released once the power supplies and internal clock circuits have stabilized. The $\overline{\text{RESET}}$ signal should not only offer a suitable delay, but it should also have a clean monotonic edge. Analog Devices has a range of microprocessor supervisory ICs with different features. Features include one or more of the following.

- Power-up reset
- Optional manual reset input
- Power low monitor
- Backup battery switching

The part number series for reset and supervisory circuits from Analog Devices are as follows.

- ADM69x
- ADM70x
- ADM80x
- ADM1232
- ADM181x
- ADM869x

A simple power-up reset circuit is shown in Figure 14-6 using the ADM809-RART reset generator. The ADM809 provides an active low $\overline{\text{RESET}}$ signal whenever the supply voltage is below 2.63 V. At power-up, a 240 ms active reset delay is generated to give the power supplies and oscillators time to stabilize.

Another part, the ADM706TAR, provides power on $\overline{\text{RESET}}$ and optional manual $\overline{\text{RESET}}$. It allows designers to create a more complete supervisory circuit that monitors the supply voltage. Monitoring the supply voltage allows the system to initiate an orderly shutdown in the event of power failure. The ADM706TAR also allows designers to create a watchdog timer that monitors for software failure. This part is available in an 8-lead SOIC package. Figure 14-7 shows a typical application circuit using the ADM706TAR.

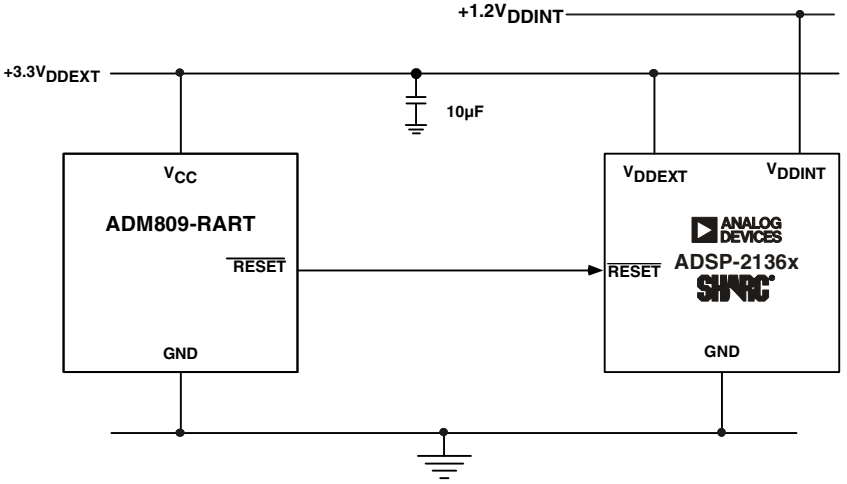


Figure 14-6. Simple Reset Generator

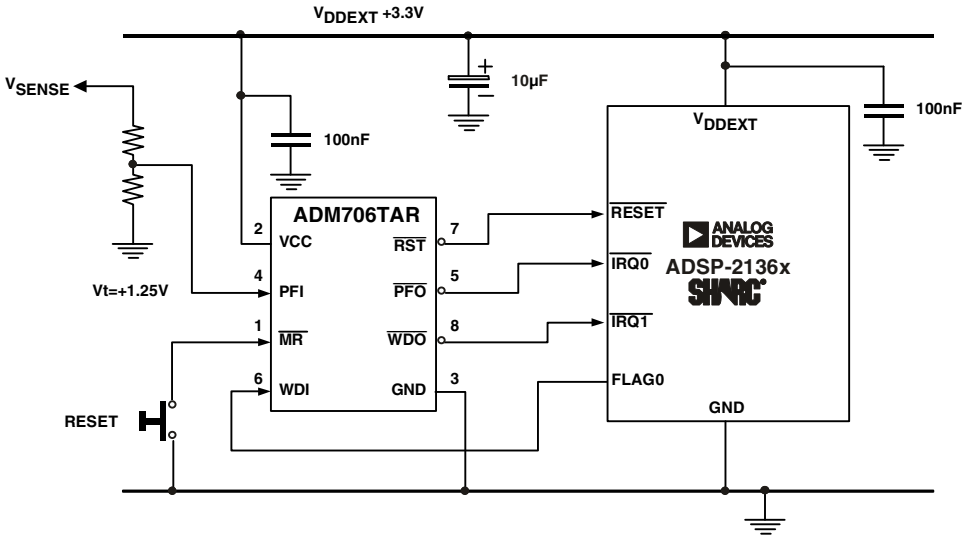


Figure 14-7. Reset Generator and Power Supply Monitor

Designing for High Frequency Operation

Because the processor must be able to operate at very high clock frequencies, signal integrity and noise problems must be considered for circuit board design and layout. The following sections discuss these topics and suggest various techniques to use when designing and debugging target systems.

All synchronous processor behavior is specified to `CLKIN`. System designers are encouraged to clock synchronous peripherals with this same clock source (or a different low-skew output from the same clock driver).

Other Recommendations and Suggestions

- Use more than one ground plane on the PCB to reduce crosstalk. Be sure to use lots of vias between the ground planes. One V_{DD} plane for each supply is sufficient. These planes should be in the center of the PCB.
- To reduce crosstalk, keep critical signals such as clocks, strobes, and bus requests on a signal layer next to a ground plane away from, or lay out these signals perpendicular to, other non-critical signals.
- If possible, position the processors on both sides of the board to reduce area and distances.
- To allow better control of impedance and delay, and to reduce crosstalk, design for lower transmission line impedances.
- Use 3.3 V peripheral components and power supplies to help reduce transmission line problems, ground bounce, and noise coupling (the receiver switching voltage of 1.5 V is close to the middle of the voltage swing).
- Experiment with the board and isolate crosstalk and noise issues from reflection issues. This can be done by driving a signal wire from a pulse generator and studying the reflections while other components and signals are passive.

Decoupling and Grounding

Extended copper planes must be used for the ground and power supplies. Designs should use an absolute minimum of 12 bypass capacitors (four 0.1 µF, four 10 nF and four 1nF ceramic) for each VDDEXT and VDDINT supply. More extensive bypassing may be required for some applications. The capacitors should be placed close to the package as shown in Figure 14-8. The decoupling capacitors should be tied directly to the

power and ground planes with vias that touch their solder pads. Surface-mount capacitors are recommended because of their lower series inductances (ESL) and higher series resonant frequencies. Connect the power and ground planes to the ADSP-2136x processor's power supply pins directly with vias—do not use traces. The ground planes should not be densely perforated with vias or traces as this reduces their effectiveness. In addition, there should be several large tantalum capacitors on the board.

Designs can use either bypass placement case shown in Figure 14-8, or combinations of the two. Designs should try to minimize signal feedthroughs that perforate the ground plane.

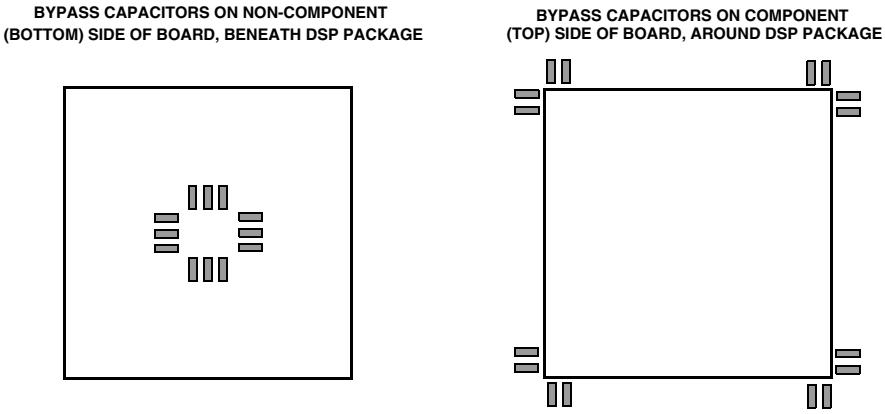

BYPASS CAPACITORS ON NON-COMPONENT
(BOTTOM) SIDE OF BOARD, BENEATH DSP PACKAGE
BYPASS CAPACITORS ON COMPONENT
(TOP) SIDE OF BOARD, AROUND DSP PACKAGE

Figure 14-8. Bypass Capacitor Placement

Oscilloscope Probes

When making high speed measurements, be sure to use a "bayonet" type or similarly short (< 0.5 inch) ground clip, attached to the tip of the oscilloscope probe. The probe should be a low capacitance active probe with 3 pF or less of loading. The use of a standard ground clip with four inches of ground lead causes ringing to be seen on the displayed trace and makes the signal appear to have excessive overshoot and undershoot.

A 1 GHz or better sampling oscilloscope is needed to see the signals accurately.

Recommended Reading

The text *High-Speed Digital Design: A Handbook of Black Magic* is recommended for further reading. This book is a technical reference that covers the problems encountered in state-of-the-art, high frequency digital circuit design. It is also an excellent source of information and practical ideas. Topics covered in the book include:

- High-Speed Properties of Logic Gates
- Measurement Techniques
- Transmission Lines
- Ground Planes and Layer Stacking
- Terminations
- Vias
- Power Systems
- Connectors
- Ribbon Cables
- Clock Distribution
- Clock Oscillators

High-Speed Digital Design: A Handbook of Black Magic, Johnson & Graham, Prentice Hall, Inc., ISBN 0-13-395724-1.

High-Speed Signal Propagation: Advanced Black Magic, Johnson & Graham, Prentice Hall, Inc., ISBN 0-13-084408-X.

Processor Booting

When a processor is initially powered up, its internal SRAM is undefined. Before actual program execution can begin, the application must be loaded from an external non-volatile source such as flash memory or a host processor. This process is known as *bootstrap loading* or *booting* and is automatically performed by the processor after power-up or after a software reset.

Boot Mechanisms

In order to ensure proper device booting, the following hardware mechanisms are available on the processor.

- Peripheral boot configuration pins
- Peripheral control settings
- Peripheral DMA parameter settings
- Core IDLE (held in reset during boot kernel load)
- Peripheral and global interrupts (jump to DMA interrupt)
- Hardware and software resets to initiate the boot process

Booting Process

Each boot mode has a process it uses to load initial code into the processor.

For this chapter, the interrupt vector table addresses are defined as: `IVT_Start_Addr` = 0x90000 and `IVT_End_Addr` = 0x900FF.

Loading the Boot Kernel Using DMA

1. At reset, the processor is hardwired (using the boot configuration pins) to load 256 x 48-bit instruction words via a DMA starting at IVT_START_ADDRESS.
2. The sequencer will halt into $\overline{\text{RESET}}$ location until an interrupt occurs.

Executing the Boot Kernel

1. The DMA completes (counter zero) and the interrupt associated with the peripheral that the processor is booting from is activated.
2. The processor jumps to the applicable interrupt vector location and executes the code located there.

Typically, the first instruction at the interrupt vector is a return from interrupt (RTI) instruction.

If you write your own loader kernel, you must ensure that the first instruction (boot kernel) in the appropriate peripheral is an RTI instruction.

Loading the Application

1. Once the kernel is executed, the application code is booted. by the kernel reading the header from the boot stream and decoding it.
2. The loader kernel executes a series of direct memory accesses (DMAs) to import the rest of the application depending on data sizes and code blocks.

Loading the Application's Interrupt Vector Table

1. The last header is recognized by the kernel indicating that booting has nearly finished.
2. The kernel prepares a 256 x 48-word DMA starting at `IVT_START_ADDRESS`.

   This overrides the kernel with the application's IVT. However, the elfloader needs to temporarily include the RTI instruction at the peripheral interrupt address, allowing a return from the last interrupt.
3. The RTI instruction overrides the address where the code line is stored.

While both DMA types ("Loading the Boot Kernel Using DMA" and "Loading the Application's Interrupt Vector Table") seem similar, loading the kernel is accomplished using hardware while loading the IVT is accomplished using software.

It is very important to match the dedicated kernel to the dedicated boot type (for example SPI kernel and SPI boot type) in the elfloader property page. If this is not done, the RTI instruction (in "Loading the Application's Interrupt Vector Table") will not be placed at the correct address. This causes execution errors.

Starting Program Execution

The processed interrupt returns the sequencer to the reset location by performing the two following steps

1. Overriding the RTI instruction with the original code line
2. Starting program execution from reset location

Internal Memory Kernel Load

The internal memory blocks of SHARC processors are based on 4 columns, each of which is 16-bits in I/O width. The access type is defined on the address space (for example short word address space only performs a one column access). Different addressing modes result in different address counts (for example one long word address represents four short word addresses). The four different access types are:

- 1-column access (short word, 16-bit)
- 2-column access (normal word, 32-bit)
- 3-column access (normal word, 48-bit)
- 4-column access (long word, 64-bit)

During the boot process, word packing of 8 to 32-bit is performed. In other words, the kernel is not loaded directly with 256 x 48-bit words, instead it is loaded with 384 x 32-bit 'packed words' (2-column access). Note that the sequencer considers the IVT as opcode and fetches the block in 3-column mode, (or 48-bit instructions).

The same physical IVT space is booted by DMA in 2 column (IVT_START_ADDR – (IVT_START_ADDR + 0x17F)) and fetched by the sequencer in 3-column (IVT_START_ADDR – IVT_END_ADDR). For more information about 32- and 48-bit internal memory addressing, see the "Memory" chapter in the *SHARC Processor Programming Reference*.

For the ADSP-2136x SHARC processors IVT_START_ADDR = 0x90000 and IVT_END_ADDR = 0x900FF.

Parallel Port Booting

The SHARC processors support an 8-bit boot mode through the parallel port. The processor is configured for 8-bit boot mode when the BOOT_CFG1-0 pins = 10. For a complete diagram of the parallel port, see Figure 4-1 on page 4-3.

After $\overline{\text{RESETOUT}}$ de-asserted, the processor starts to drive:

- CS (ADDR[23:16]) to select EPROM/FLASH
- ALE driven low
- AD[7:0] from multiplexed address/data bus
- $\overline{\text{RD}}$ strobe with 23 PCLK cycle wait states

Since ALE polarity is active high by default, systems using parallel port boot mode must use address latching hardware that can process this active high signal.

Switching from address to data transfer (muxed bus) the parallel port latches 8-bit words in the receive shift buffer PPSI. Packing of 4 x 8 words to a 32-bit word least significant bit (LSB) first is performed and is shifted to the receive buffer RXPP which finally transfers by DMA into internal memory (Figure 14-9).

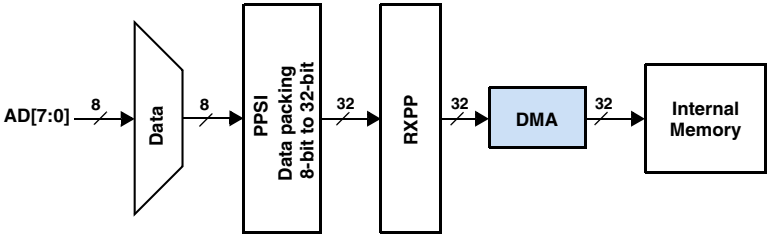


Figure 14-9. Parallel Port Data Packing

Unlike previous SHARCs, the 8 to 48-bit packing mode for instructions is not supported. For instructions, the DMA reads 3 x 32-bit data which results in 2 x 48-bit instructions.

Also unlike previous SHARC processors, the ADSP-21362/3/4/5/6 processors do not have a boot memory select (BMS) pin. The boot FLASH/EPROM's chip select (CS) should be generated from an address decoder, or otherwise derived from the parallel port signals. For more information, see Chapter 4, Parallel Port.

The parallel port bits used in booting are shown in Table 14-10. For a complete description of the parallel port control register, see "Parallel Port Control Register (PPCTL)" on page A-11. Note that after reset, the value of this register has changed to 0x0000402E in no boot mode (reserved).

Table 14-10. PPCTL Boot Settings (0x412F)

Bit	Setting
PPEN (bit 0)	= 1; enable parallel port
PPDUR (bits 5–1)	= 10111; (23 core clock cycles per data transfer cycle)
PPBHC (bit 6)	= 0; do not insert a bus hold cycle on every access
PP16 (bit 7)	= 0; external data width = 8 bits
PPDEN (bit 8)	= 1; use DMA
PPTRAN (bit 9)	= 0; receive (read) DMA
PPBHD (bit 12)	= 0; buffer hang enabled
PPALEPL (bit 13)	= 0; ALE is active high
PPFLMD (bit 14)	= 1; enable flash mode

The parallel port DMA channel is usedwhen downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in Table 14-11.

In this configuration, the loader kernel is read via DMA from the FLASH. If the application needs to speed-up read accesses, programs should

change the duration cycles (PPDUR bits, see Table 14-10) in the kernel file. After the kernel is executed, the new duration cycle settings are applied and processor booting continues.

Table 14-11. Parameter Initialization for Parallel Port Boot

Parameter Register	Initialization Value	Comment
IIPP	IVT_START_ADDR	Start block 0
ICPP	0x180	384 x 32-bit words.
IMPP	0x01	
EIPP	0x0	External byte address
ECPP	0x600	1536 x 8-bit words
EMPP	0x1	

SPI Port Booting

The SHARC processors support booting from a host processor using SPI slave mode (BOOT_CFG1-0 = 00). Booting from an SPI flash, SPI PROM, or a host processor via SPI master mode (BOOT_CFG1-0 = 01). Both SPI boot modes (master and slave) support 8-, 16-, or 32-bit SPI devices.

In both (master and slave) boot modes, the SPI mode 3 is selected (clock polarity and clock phase = 1).

Master Boot Mode

In master boot mode, the processor initiates the booting operation by:

1. Activating the SPICLK signal and asserting the FLAG0 signal to the active low state.
2. Writing the read command 0x03 and address 0x00 to the slave device as shown in Figure 14-11.

Master boot mode is used when the processor is booting from an SPI-compatible serial PROM, serial FLASH, or slave host processor. The specifics of booting from these devices are discussed individually. On reset, the interface starts up in master mode performing a 384 32-bit word DMA transfer.

SPI master booting uses the default bit settings shown in Figure 14-12.

Table 14-12. SPICTL Master Boot Settings (0x5D06)

Bit	Setting	Comment
SPIEN	Set (= 1)	SPI enabled
SPIMS	Set (= 1)	Master device
MSBF	Cleared (= 0)	LSB first
WL	10	32-bit SPI receive shift register word length
DMISO	Cleared (= 0)	MISO enabled
SENDZ	Set (= 1)	Send zeros
SPIRCV	Set (= 1)	Receive DMA enabled
CLKPL	Set (= 1)	Active low SPI clock
CPHASE	Set (= 1)	Toggle SPICLK at the beginning of the first bit

The SPI DMA channel is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in Figure 14-11.

Table 14-13. Parameter Initialization Values for SPI Master Boot

Parameter Register	Initialization Value	Comment
SPIBAUD	0x64	SPICLK = PCLK/100
SPIFLG	0xFE01	FLAG0 used as slave-select
SPIDMAC	0x0000 0007	Enable receive interrupt on completion
IISPI	IVT_START_ADDR	Start of block 0

Table 14-13. Parameter Initialization Values for SPI Master Boot (Cont'd)

Parameter Register	Initialization Value	Comment
IMSPI	0x1	32-bit data transfers
CSPI	0x180	384 × 32-bit transfers

Master Header Information

The transfer is initiated by the transferring the necessary header information on the interface (consisting of the read opcode and the starting address of the block to be transferred, which is usually all zeros). The read opcode is fixed as 0xC0 (LSBF format) and is 24-bits long. The 8-bits that are received following the read opcode should be programmed as 0xA5 (see Figure 14-10). If the 8-bits are not programmed as 0xA5 the master boot transfer is aborted. The transfer continues until 384 x 32-bit words have been transferred, which may correspond to the loader program (just as in the slave boot mode).

1. Default state of SPICLK signal high (out of reset).
2. De-asserting the FLAG0 signal (chip select) to the active low state and toggling the SPICLK signal.
3. Reading the read command 0x03 (MSBF format to match the LSBF format) and address 0x00 from the slave device.

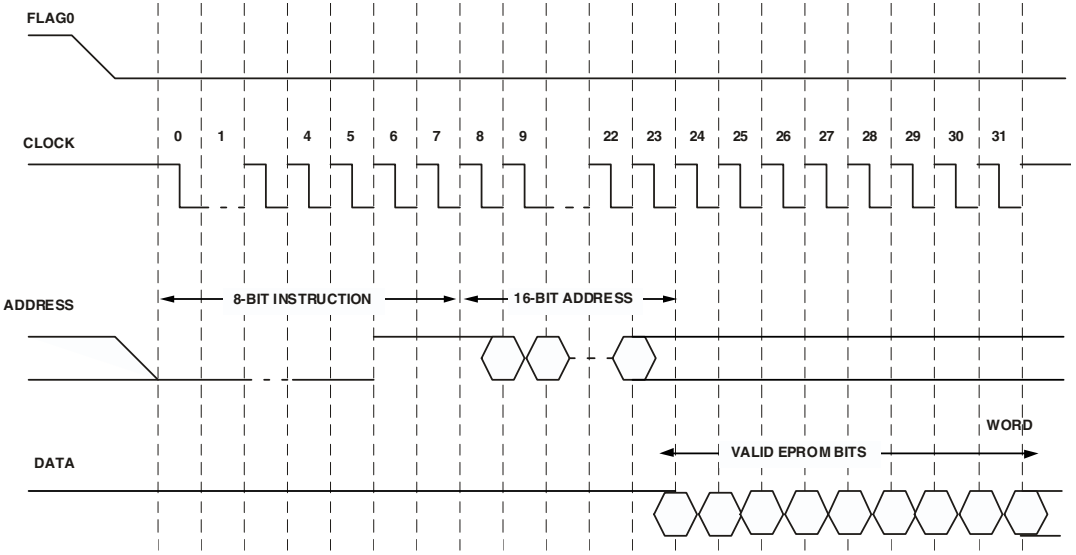


Figure 14-10. SPI Master Mode Booting Using Various Serial Devices

Slave Boot Mode

In slave boot mode, the host processor initiates the booting operation by activating the SPICLK signal and asserting the SPIDS signal to the active low state. The 256-word kernel is loaded 32 bits at a time, through the SPI receive shift register (RXSR). To receive 256 instructions (48-bit words) properly, the SPI DMA initially loads a DMA count of 0x180 (384) 32-bit words, which is equivalent to 0x100 (256) 48-bit words.

The processor's SPIDS pin should be populated with a pull-up resistor to ensure high level after power-up. When in SPI slave mode, including booting, the SPIDS signal is required to transition from high to low. SPI slave booting usesthe default bit settings shown in Table 14-14.

Table 14-14. SPI Slave Boot Bit Settings (0x4D22)

Bit	Setting	Comment
SPIEN	Set (= 1)	SPI enabled
SPIMS	Cleared (= 0)	Slave device
MSBF	Cleared (= 0)	LSB first
WL	10, 32-bit SPI	Receive shift register word length
DMISO	Set (= 1) MISO	MISO disabled
SENDZ	Cleared (= 0)	Send last word
SPIRCV	Set (= 1)	Receive DMA enabled
CLKPL	Set (= 1)	Active low SPI clock
CPHASE	Set (= 1)	Toggle SPICLK at the beginning of the first bit

The SPI DMA channel is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in Table 14-15.

Table 14-15. Parameter Initialization Value for SPI Slave Boot

Parameter Register	Initialization Value	Comment
SPIDMAC	0x0000 0007	Enable receive, interrupt on completion
IISPI	IVT_START_ADDR	Start of block 0
IMSPI	0x1	32-bit data transfers
CSPI	0x180	384 × 32-bit transfers

SPI Boot Packing

In all SPI boot modes, the data word size in the shift register is hardwired to 32 bits. Therefore, for 8- or 16-bit devices, data words are packed into the shift register to generate 32-bit words least significant bit (LSB) first, which are then shifted into internal memory. The relationship between

the 32-bit words received into the RXSPI register and the instructions that need to be placed in internal memory is shown in the following sections.

For more information about 32- and 48-bit internal memory addressing, see the "Memory" chapter in the *SHARC Processor Programming Reference*.

As shown in Figure 14-11, two words shift into the 32-bit receive shift register (RXSR) before a DMA transfer to internal memory occurs for 16-bit SPI devices. For 8-bit SPI devices, four words shift into the 32-bit receive shift register before a DMA transfer to internal memory occurs.

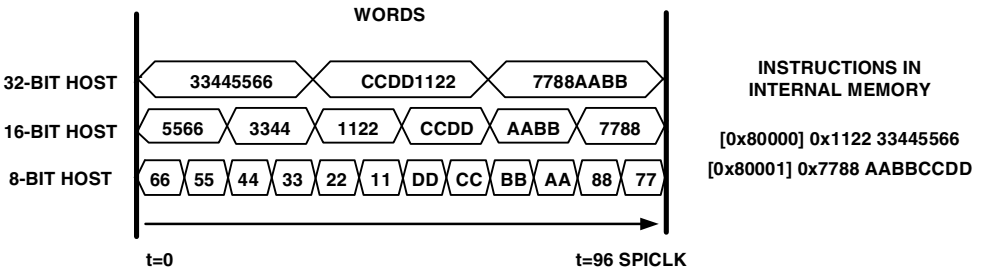


Figure 14-11. Instruction Packing for Different Hosts

When booting, the ADSP-2136x processors expect to receive words into the RXSPI register seamlessly. This means that bits are received continuously without breaks. For more information, see "Core Transfers" on page 7-22. For different SPI host sizes, the processor expects to receive instructions and data packed in a least significant word (LSW) format.

Figure 14-11 shows how a pair of instructions are packed for SPI booting using a 32-, 16-, and an 8-bit device. These two instructions are received as three 32-bit words.

The following sections examine how data is packed into internal memory during SPI booting for SPI devices with widths of 32, 16, or 8 bits.

32-Bit SPI Packing

Figure 14-12 shows how a 32-bit SPI host packs 48-bit instructions executed at PM addresses 0x90000 and 0x90001. The 32-bit word is shifted to internal program memory during the 256-word kernel load.

The following example shows a 48-bit instruction executed:

```
[0x90000] 0x112233445566
[0x90001] 0x7788AABBCCDD
```

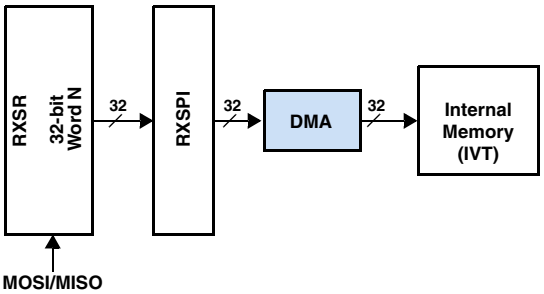


Figure 14-12. 32-Bit SPI Master/Slave Packing

The 32-bit SPI host packs or prearranges the data as:

SPI word 1= 0x33445566
SPI word 2 = 0xCCDD1122
SPI word 3 = 0x7788AABB

The initial boot of the 256-word loader kernel requires a 32-bit host to transmit 384 x 32-bit words. The SPI DMA count value of 0x180 is equal to 384 words.

16-Bit SPI Packing

Figure 14-13shows how a 16-bit SPI host packs 48-bit instructions at PM addresses 0x90000 and 0x90001. For 16-bit hosts, two 16-bit words are packed into the shift register to generate a 32-bit word. The 32-bit word shifts to internal program memory during the kernel load.

The following code shows a 48-bit instruction executed.

```
[0x90000] 0x112233445566
[0x90001] 0x7788AABBCCDD
```

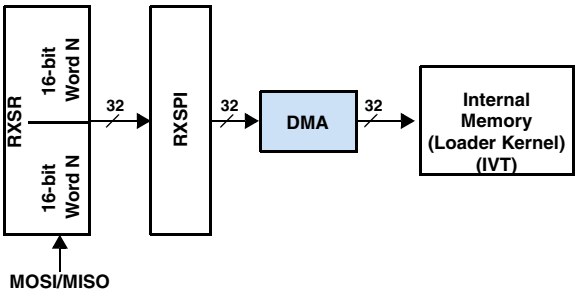


Figure 14-13. 16-Bit SPI Master/Slave Packing

The 16-bit SPI host packs or prearranges the data as:

SPI word 1 = 0x5566
SPI word 2 = 0x3344
SPI word 3 = 0x1122
SPI word 4 = 0xCCDD
SPI word 5 = 0xAABB
SPI word 6 = 0x7788

The initial boot of the 256-word loader kernel requires a 16-bit host to transmit 768 16-bit words. Two packed 16-bit words comprise the 32-bit word. The SPI DMA count value of 0x180 is equivalent to 384 words. Therefore, the total number of 16-bit words loaded is 768.

8-Bit SPI Packing

Figure 14-14 shows how an 8-bit SPI host packs 48-bit instructions executed at PM addresses 0x90000 and 0x90001. For 8-bit hosts, four 8-bit words pack into the shift register to generate a 32-bit word. The 32-bit word shifts to internal program memory during the load of the 256-instruction word kernel.

The following code shows a 48-bit instruction executed:

```
[0x90000] 0x112233445566
[0x90001] 0x7788AABBCCDD
```

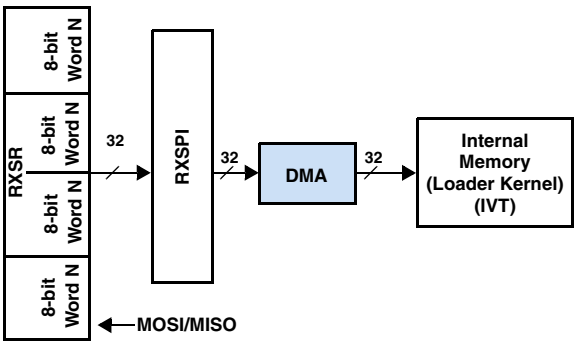


Figure 14-14. 8-Bit SPI Slave Packing

The 8-bit SPI host packs or prearranges the data as:

SPI word 1 =	0x66	SPI word 7 =	0xDD
SPI word 2 =	0x55	SPI word 8 =	0xCC
SPI word 3 =	0x44	SPI word 9 =	0xBB
SPI word 4 =	0x33	SPI word 10 =	0xAA
SPI word 5 =	0x22	SPI word 11 =	0x88
SPI word 6=	0x11	SPI word 12=	0x77

The initial boot of the 256-word loader kernel requires an 8-bit host to transmit 1536 x 8-bit words. The SPI DMA count value of 0x180 is equal

to 384 words. Since one 32-bit word is created from four packed 8-bit words, the total number of 8-bit words transmitted is 1536.

Kernel Boot Time

This section and Table 14-16 describe the minimum required booting time for the kernels (provided by the tools). There are 5 timing windows which together describe the entire boot process.

1. $\overline{\texttt{RESET}}$ to $\overline{\texttt{RESETOUT}}$ (core is in reset)
2. $\overline{\texttt{RESETOUT}}$ to chip select boot source (activate the boot DMA)
3. Load Kernel DMA (256 words)
4. Load application (user dependent)
5. Load IVT (256 words)

For SPI slave boot, SPIDS should only be asserted after $\overline{\texttt{RESETOUT}}$ has de-asserted.

Table 14-16. Kernel Boot Time

Boot Mode	$\overline{\texttt{RESET}}$ to $\overline{\texttt{RESETOUT}}$	$\overline{\texttt{RESETOUT}}$ to Boot Chip Select	Kernal DMA (256 Words)	Comment
SPI Master	4096 CLKIN	1 PCLK	(I/O × PCLK ÷ 100 + 4 × PCLK) × N	N=384, 768 or 1536 for I/O=32, 16 or 8
SPI Slave	4096 CLKIN	Host drives signal	(I/O × PCLK ÷ 100 + 2 × PCLK) × N	N=384, 768 or 1536 for I/O=32, 16 or 8
Parallel Port	4096 CLKIN	9 PCLK	28 × PCLK × 1536	8-bit access

The complete time for booting can be estimated by adding all 5 timing windows. Loading Kernel and Loading IVT both have the same size, however the default access time (wait states) for the IVT loading can be changed in the kernel by the user.

Definition of Terms

Booting

When a processor is initially powered up, its internal SRAM and many other registers are undefined. Before actual program execution can begin, the application must be loaded from an external non-volatile source such as flash memory or a host processor. This process is known as bootstrap loading or booting and is automatically performed by the processor after power-up or after a software reset.

Boot Compression

Boot compression is only supported by software. The loader utility takes two steps to compress a boot stream.

1. The loader generates the boot stream in the conventional way (builds data blocks).
2. The loader then applies compression to the boot stream.

Decompression is the reverse of this process.

1. The loader utility decompresses the compressed stream.
2. The loader then loads code and data into memory segments in the conventional way.

For more information on elfloader compression refer to the VisualDSP tools documentation.

Boot Kernel

The boot kernel is an executable file which schedules the entire boot process. The temporary location of the kernel resides in the processor's Interrupt vector location (IVT). The IVT typically has a maximum size of 256 x 48 words. After booting, the kernel overwrites this area.

These kernel files (DXE, ASM) are supplied with the VisualDSP++ development tools for all boot modes. For more information on the kernels, refer to the tools documentation

Boot Master/Slave

How a processor boots is dependent on the peripheral used. In master mode, the processor drives all signals to the external device (for example the SPI chip select signal, `CS` or MISO). In this mode, the processor has full control over the boot process. In slave mode, the processor expects data to be driven from the external master or host at a specific time (for example the SPI device select signal, `SPIDS` or MOSI. In this mode, the processor has only partial control over the boot process.

Boot Modes

The boot mode is identified by the `BOOT_CFG1-0` pins that are used in the boot process.

Elfloader

The elfloader is a tool that converts an executable image (`.dxe` file) into a boot stream (`.ldr` file). During this process the elfloader performs operations that remove redundant information (like symbols), or adds header information into the boot stream. This information is decoded by the loader kernel and is required to schedule boot scenario. For more information on the elfloader, refer to the VisualDSP tools documentation

Elfsplitter

The elfsplitter is a tool used for no boot mode. The elfsplitter converts an executable image (`.dxe` file) into a non boot stream (`.ldr` file). For more information on the elfsplitter, refer to the VisualDSP tools documentation. The SHARC processors do not support no boot mode.

No Boot Mode

In this mode, the processor does not boot. Instead, it starts fetching instructions directly from external memory. The SHARC processors do not support this mode.

Reserved Boot Mode

For `BOOT_CFG1-0` pins =11, the processor does not boot. Instead, it starts fetching instructions directly from the internal ROM. Only specific versions of the ADSP-21362/3/4/5/6 processors support this mode.

A REGISTERS REFERENCE

The ADSP-21362/3/4/5/6 processors have general-purpose and dedicated registers in each of their functional blocks. The register reference information for each functional block includes bit definitions, initialization values, and memory-mapped addresses (for I/O processor registers). Information on each type of register is available at the following locations:

- "I/O Processor Registers" on page A-2
- "Power Management Control Register (PMCTL)" on page A-6
- "Parallel Port Registers" on page A-10
- "Serial Peripheral Interface Registers" on page A-14
- "Pulse Width Modulation Registers" on page A-23
- "Serial Port Registers" on page A-30
- "Input Data Port Registers" on page A-47
- "Peripheral Timer Registers" on page A-56
- "Sample Rate Converter Registers" on page A-59
- "Precision Clock Generator Registers" on page A-65
- "Sony/Philips Digital Interface Registers" on page A-70
- "DAI Interrupt Controller Registers" on page A-77
- "DAI Status Register" on page A-78

- "DAI Signal Routing Unit Registers" on page A-80
- "Register Listing" on page A-108

When writing programs, it is often necessary to set, clear, or test bits in the processor's registers. While these bit operations can all be done by referring to the bit's location within a register or (for some operations) the register's address with a hexadecimal number, it is much easier to use symbols that correspond to the bit's or register's name. For convenience and consistency, Analog Devices supplies a header file that provides these bit and registers definitions. An #include file is provided with VisualDSP++ tools and can be found in the `VisualDSP/2136x/include` directory.

Many registers have reserved bits. When writing to a register, programs may only clear (write zero to) the register's reserved bits.

I/O Processor Registers

The I/O processor's registers are accessible as part of the processor's memory map. "Register Listing" on page A-108 lists the I/O processor's memory-mapped registers and provides address reset values. These registers occupy addresses 0x0000 0000 through 0x0003 FFFF of the memory map.

Since the I/O processor registers are memory-mapped, the processor's architecture does not allow programs to directly transfer data between these registers and other memory locations, except as part of a DMA operation. To read or write I/O processor registers, programs must use the processor core registers.

The register names for I/O processor registers are not part of the processor's assembly syntax. To ease access to these registers, programs should use the header file containing the registers' symbolic names and addresses.

Notes on Reading Register Drawings

The register drawings in this appendix provide "at-a-glance" information about specific registers. They are designed to give experienced users basic information about a register and its bit settings. When using these registers, the following should be noted.

1. The figures provide the bit mnemonic and its definition. Where necessary, detailed descriptions can be found in the tables that follow the register drawings and in the chapters that describe the particular module.
2. The VisualDSP++ tools suite contains the complete listing of registers in a header file, `def2136x.h`.
3. "Register Listing" on page A-108 provides a complete list of user accessible registers, their addresses, and their state at reset.
4. In most cases, control registers are read/write (RW) and status registers are read only (RO). Some registers provide error status bits which are write-one-to-clear (W1C). Where individual bits within a register differ, they are noted in the register drawing.

System Control Register (SYSCTL)

The SYSCTL register is used to set up system configuration selections. Bit settings for this register are shown in Figure A-1 and described in Table A-1. The reset value has all bits initialized to zero.

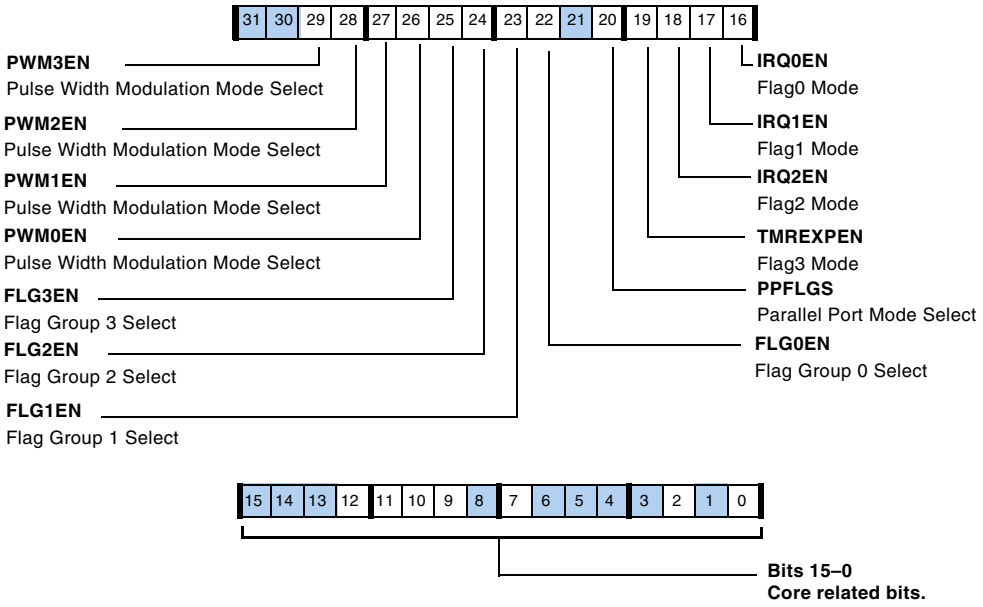


Figure A-1. SYSCTL Register

Table A-1. SYSCTL Register

Bit	Name	Description
15–0	Core related bits. For information on these bits, see the *SHARC Processor Programming Reference*.	
16	IRQ0EN	**Flag0 Interrupt Mode.** 0 = FLAG0 pin is a general-purpose I/O pin. Permits core writes. 1 = FLAG0 pin is allocated to interrupt request $\overline{\text{IRQ0}}$.

Table A-1. SYSCTL Register (Cont'd)

Bit	Name	Description
17	IRQ1EN	**Flag1 Interrupt Mode.** 0 = FLAG1 pin is a general-purpose I/O pin. Permits core writes. 1 = FLAG1 pin is allocated to interrupt request $\overline{\texttt{IRQ1}}$.
18	IRQ2EN	**Flag2 Interrupt Mode.** 0 = FLAG2 pin is a general-purpose I/O pin. Permits core writes. 1 = FLAG2 pin is allocated to interrupt request $\overline{\texttt{IRQ2}}$.
19	TMREXPEN	**Core Timer Expired.** Read/Write 0 = FLAG3 pin is a general-purpose I/O pin. Permits core writes. 1 = FLAG3 pin output is timer expired signal (TMREXP).
20	PPFLGS	**Parallel Port Mux Select.** 0 = Parallel port mux enabled 1 = Parallel port mux disabled. When used in conjunction with the FLGxEN bits, the parallel port AD15–0 pins are in flag mode. Permits core writes. Configuring the parallel port pins to function as FLAG0–15 also causes the FLAG0–3 pins to change to their alternate role, $\overline{\texttt{IRQ0-2}}$ and TMREXP.
21	Reserved	
22	FLG0EN	**Flag Group 0 Select.** 0 = AD11–8 pins in parallel port mode 1= AD11–8 pins in flag mode (FLG3–0)
23	FLG1EN	**Flag Group 1 Select.** 0 = AD15–12 pins in parallel port mode 1 = AD15–12 pins in flag mode (FLG7–4)
24	FLG2EN	**Flag Group 2 Select.** 0 = AD3–0 pins in parallel port mode 1 = AD3–0 pins in flag mode (FLG11–8)
25	FLG3EN	**Flag Group 3 Select.** 0 = AD7–4 pins in parallel port mode 1 = AD7–4 pins in flag mode (FLG15–12)
26	PWM0EN	**Pulse Width Modulation0 Mode Select.** 0 = AD11–8 pins in parallel port mode 1 = AD11–8 pins in PWM mode

Table A-1. SYSCTL Register (Cont'd)

Bit	Name	Description
27	PWM1EN	**Pulse Width Modulation1 Mode Select.** 0 = AD15–12 pins in parallel port mode 1 = AD15–12 pins in PWM mode
28	PWM2EN	**Pulse Width Modulation2 Mode Select.** 0 = AD3–0 pins in parallel port mode 1 = AD3–0 pins in PWM mode
29	PWM3EN	**Pulse Width Modulation3 Mode Select.** 0 = AD7–4 pins in parallel port mode 1 = AD7–4 pins in PWM mode
31–30	Reserved	

Power Management Control Register (PMCTL)

The following sections describe the registers associated with the processors power management functions.

The power management control register, shown in Figure A-2, is a 32-bit memory-mapped register. This register contains bits to control phase lock loop (PLL) multiplier and divider (both input and output) values, PLL bypass mode, and clock enabling control for peripherals (see Table A-2 on page A-8). This register also contains status bits, which keep track of the status of the `CLK_CFG` pins (read-only). The reset value of `PMCTL` is dependent on the `CLK_CFG` pins (bits 5–0).

The core can write to all bits except the read-only status bits. The `DIVEN` bit is a logical bit, that is, it can be set, but on reads it always responds with zero.

When the PLL is programmed using a multiplier and a divisor, the DIVEN and PLLBP bits should NOT be programmed in the same core clock cycle. For more information, see "Bypass Clock" on page 14-10 and "Example for Output Divider Management" on page 14-13.

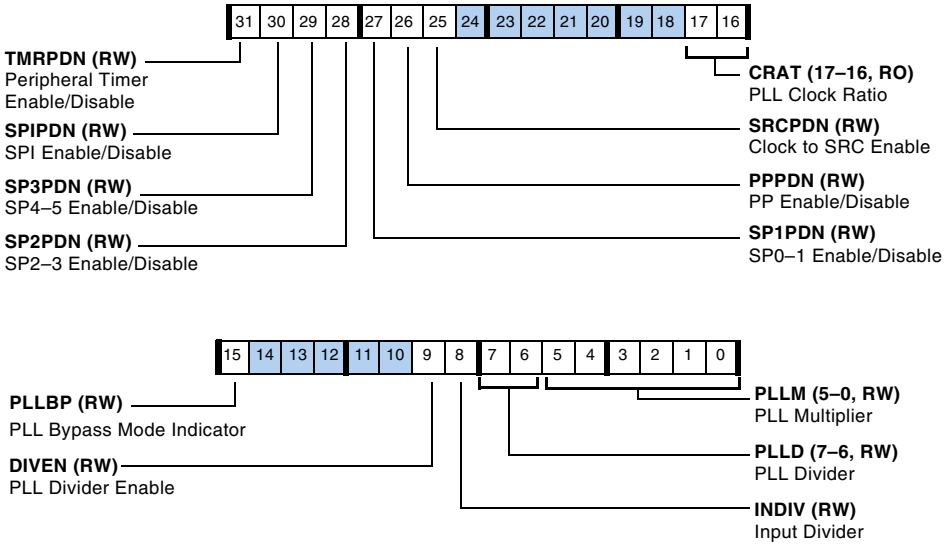


Figure A-2. PMCTL Register

Table A-2. PMCTL Register Bit Descriptions

Bit	Name	Description
5–0	PLLM	**PLL Multiplier.** Read/Write. PLLM = 0 PLL multiplier = 64 0 < PLLM < 64 PLL multiplier = PLLM Reset value = CLK_CFG1–0 00 = 000000 = 6x 01 = 100000 = 32x 10 = 010000 = 16x 11 = 000110 = 6x (Reserved)
7–6	PLLD	**PLL Output Divider.** Read/Write 00 = output divider = 1 01 = output divider = 2 10 = output divider = 4 11 = output divider = 8 Reset value = 00
8	INDIV	**Input Divisor.** Read/Write 0 = Divide by 1 1 = Divide by 2 Reset value = 0
9	DIVEN	**Enable PLL Divider Value Loading.** Read/Write 0 = Do not load PLLD 1 = Load PLLD Reset value = 0
14–10	Reserved	
15	PLLBP	**PLL Bypass Mode Indication.** Read/Write 0 = PLL is in normal mode 1 = Put PLL in bypass mode Reset value = 0
17–16	CRAT	**PLL Clock Ratio (CLKIN to CK).** Read only. Settings of the CLK_CFG pins: 00 = CLK_CFG00 01 = CLK_CFG01 10 = CLK_CFG10 11 = CLK_CFG11(reserved)
25–18	Reserved	

Table A-2. PMCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
26	PPPDN	**Parallel Port Enable/Disable.** Read/Write 0 = PP is in normal mode 1 = Shutdown clock to PP Reset value = 0
27	SP1PDN	**SPORT1 Enable/Disable.** Read/Write 0 = SP0–1 are in normal mode 1 = Shutdown clock to SP0–1 Reset value = 0
28	SP2PDN	**SPORT2 Enable/Disable.** Read/Write 0 = SP2–3 are in normal mode 1 = Shutdown clock to SP2–3 Reset value = 0
29	SP3PDN	**SPORT3 Enable/Disable.** Read/Write 0 = SP4–5 are in normal mode 1 = Shutdown clock to SP4–5 Reset value = 0
30	SPIPDN	**SPI/SPIB Enable/Disable.** Read/Write 0 = SPI is in normal mode 1 = Shutdown clock to SPI Reset value = 0
31	TMRPDN	**Peripheral Timers Enable/Disable.** Read/Write 0 = Timer is in normal mode 1 = Shutdown clock to timer Reset value = 0

Peripheral Registers

The registers in the following sections are used for the peripherals that are not routed through the signal routing unit (SRU).

MTM DMA Control (MTMCTL Register)

The memory-to-memory (MTM) DMA register (MTMCTL) allows programs to transfer blocks of 64-bit data from one internal memory location to another. This transfer method uses two DMA channels, one for reading data and one for writing data. These transfers are controlled using the MTMCTL register shown in Figure A-3.

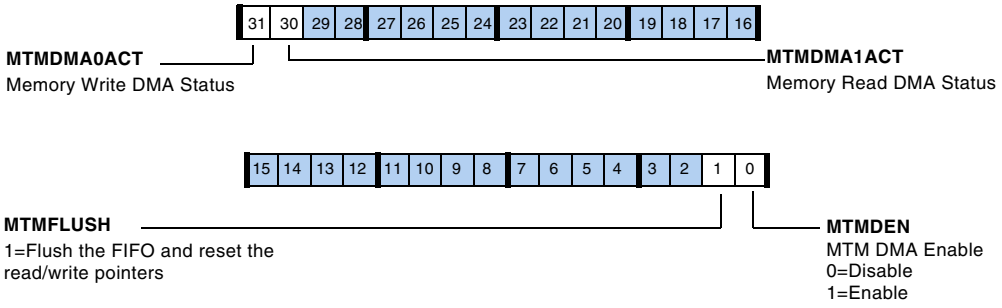

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
MTMDMA0ACT
Memory Write DMA Status
MTMDMA1ACT
Memory Read DMA Status
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
MTMFLUSH
1=Flush the FIFO and reset the read/write pointers
MTMDEN
MTM DMA Enable
0=Disable
1=Enable

Figure A-3. MTM DMA Register (MTMCTL)

Parallel Port Registers

The parallel port in the ADSP-2136x processor processor contains several user-accessible registers shown in the following sections.

Parallel Port DMA Registers

The parallel port DMA registers are described in "I/O Processor" in Chapter 2, I/O Processor.

Parallel Port Control Register (PPCTL)

The parallel port control register (PPCTL) is used to configure and enable the parallel port interface. The bit settings are shown in Figure A-4 and described in Table A-3.

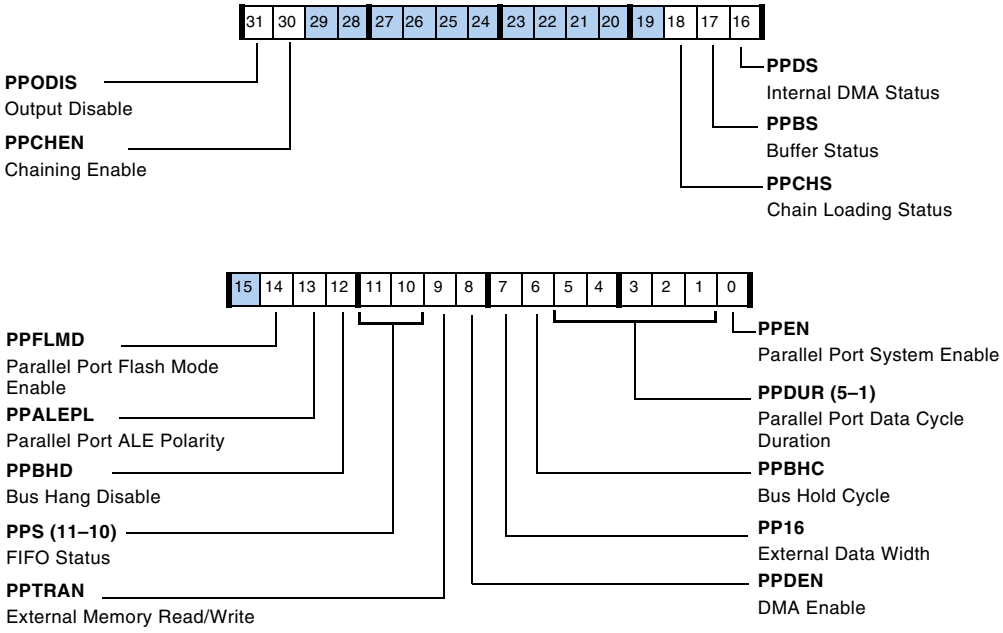


Figure A-4. PPCTL Register

Table A-3. Parallel Port Control Register (PPCTL)

Bit	Name	Description
0	PPEN	**Parallel Port Enable.** 0 = Disable parallel port Clearing this bit clears the FIFO and the parallel status information. If an $\overline{RD}$, $\overline{WR}$, or ALE cycle has already started, it completes normally before the port is disabled. The parallel port is ready to transmit or receive two cycles after it is enabled. An ALE cycle always occurs before the first read or write cycle after PPEN is enabled. 1 = Enable parallel port
5–1	PPDUR	**Parallel Port Duration.** The duration of parallel port data cycles is determined by these bits. ALE cycles are not affected by this setting and are fixed at 3 PCLK cycles. 00010 = 3 clock cycles; 55.5 MHz throughput 00011 = 4 clock cycles; 41.6 MHz throughput 00100 = 5 clock cycles; 33.3 MHz throughput 00101 = 6 clock cycles; 27.75 MHz throughput to 11111 = 32 clock cycles; 5.2 MHz throughput The default setting is for user boot mode only. Otherwise bits 1 through 5 are all zeros.
6	PPBHC	**Bus Hold Cycle.** If set (=1), this causes every data cycle to be prolonged for 1 PCLK period. If cleared (=0) no bus hold cycle occurs, and the duration of data cycle is exactly the value specified in PPDUR. Bus hold cycles do not apply to ALE cycles which are always 3 PCLK cycles. Along with the PPFLMD bit, (=1) allows user boot mode. When cleared (=0, default) for user boot mode only, otherwise 1.
7	PP16	**Parallel Port External Data Width.** 0 = External data width is 8 bits 1 = External data width is 16 bits
8	PPDEN	**Parallel Port DMA Enable.** 0 = DMA is disabled 1 = DMA enabled When PPDEN is cleared, any DMA requests already in the pipeline complete, and no new DMA requests are made. This does not affect FIFO status.
9	PPTRAN	**Parallel Port Transmit/Receive Select.** 0 = Processor is reading from external memory 1 = Processor is writing to external memory

Table A-3. Parallel Port Control Register (PPCTL) (Cont'd)

Bit	Name	Description
11–10	PPS	**Parallel Port FIFO Status.** These read-only bits indicate the status of the parallel port FIFO: 00 = RXPP/TXPP is empty 01 = RXPP/TXPP is partially full 11 = RXPP/TXPP is full
12	PPBHD	**Parallel Port Buffer Hang Disable.** 0 = Core stalls occur normally. The core stall occurs when the core attempts to write to a full transmit buffer or read from the empty receive buffer. 1 = Prevents a core hang. Old data present in the receive buffer is reread if the core tries to read it. If a write to the transmit buffer is performed, the core overwrites the current data in the buffer.
13	PPALEPL	**Parallel Port ALE Polarity Level.** 0 = ALE active high 1 = ALE active low
14	PPFLMD	**Parallel Port Flash Mode Enable.** Used to meet the slower timing requirements of typical flash memories. When set, (default, = 1) inserts a 4 peripheral clock cycle (t_{PCLK}) delay between RD rising edge and ALE rising edge (for reads). Inserts a 4 t_{PCLK} cycle delay between $\overline{WR}$ rising edge and next WR falling edge (for writes). Removes hold of address for 1 t_{PCLK} cycle after $\overline{RD}$ rising edge. When cleared (= 0) parallel port functions normally (SRAM mode).
15	Reserved	
16	PPDS	**Parallel Port DMA Status.** Read-only bit indicates: 0 = Internal DMA interface inactive 1 = Internal DMA interface active
17	PPBS	**Parallel Port Bus Status.** 0 = External bus is available 1 = External bus interface is busy. The bus is busy for the duration of the 32-bit transfer, including the ALE cycles. Note: This bit goes high two cycles after data is ready to transmit (after a data is written to PPTX, after PPRX is read with PPEN=1, after writing PPCTL to have PPEN=1 and PPDEN=1).

Table A-3. Parallel Port Control Register (PPCTL) (Cont'd)

Bit	Name	Description
18	PPCHS	**Parallel Port Chaining Status.** 0 = Chain load status not active 1 = Chain load status active. Read only.
29–19	Reserved	
30	PPCHEN	**Parallel Port Chaining Enable.** Enables DMA chaining. 0 = DMA chaining disabled 1 = DMA chaining enabled
31	PPODIS	**Output Disable.** 0 = Parallel port pins are not three-stated (disabled) 1 = All parallel port related pins, address/data and strobes are three-stated and an external agent can use the bus (enabled).

Serial Peripheral Interface Registers

The following sections describe the registers associated with the two serial peripheral interfaces (SPIs). Note that the SPI B port is routed through the DAI.

SPI Control Registers (SPICTL, SPICTLB)

The SPI control (`SPICTL`) registers are used to configure and enable the SPI system. The bit settings for these registers are shown in Figure A-5 and described in Table A-4.

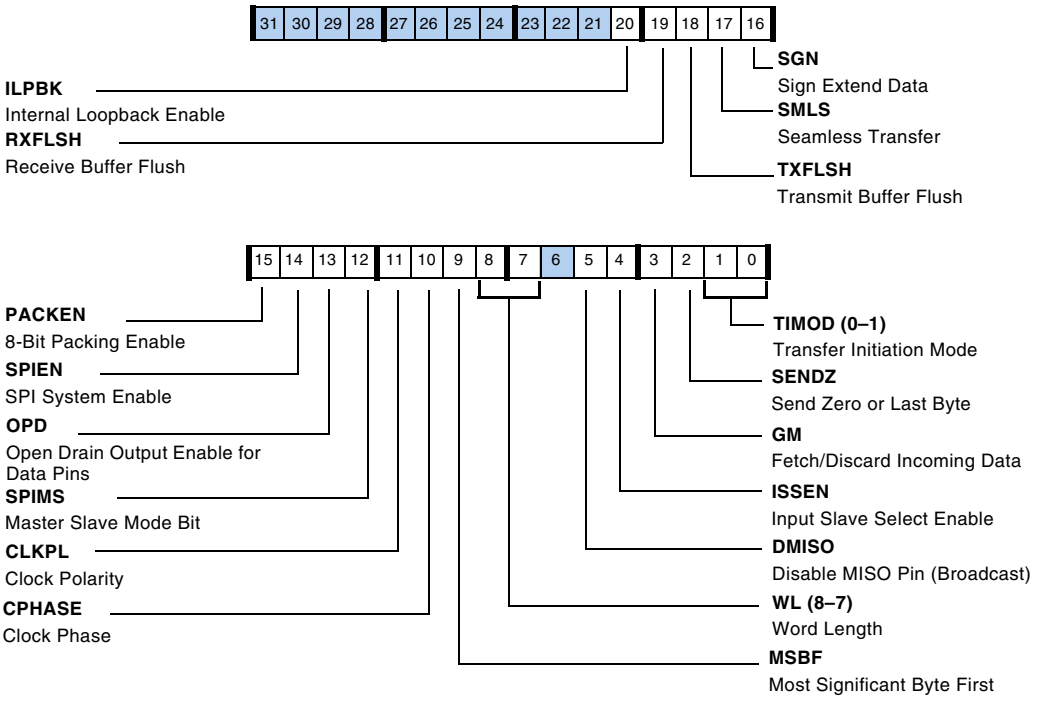


Figure A-5. SPICTL, SPICTLB Registers

Table A-4. SPICTL Register Bit Descriptions

Bit	Name	Description
1–0	TIMOD	**Transfer Initiation Mode.** Defines transfer initiation mode and interrupt generation. 00 = Initiate transfer by read of receive buffer. Interrupt active when receive buffer is full. 01 = Initiate transfer by write to transmit buffer. Interrupt active when transmit buffer is empty. 10 = Enable DMA transfer mode. Interrupt configured by DMA. 11 = Reserved
2	SENDZ	**Send Zero.** Send zero or the last word when TXSPI is empty. 0 = Send last word 1 = Send zeros

Table A-4. SPICTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
3	GM	**Get Data.** When RXSPI is full, get data or discard incoming data. 0 = Discard incoming data 1 = Get more data, overwrites the previous data
4	ISSEN	**Input Slave Select Enable.** When enabled as a master, $\overline{\text{SPIDS}}$ can serve as an error-detection input for the SPI in a multi-master environment. When this bit is set and another SPI device asserts the $\overline{\text{SPIDS}}$ signal of the current SPI master then the current master releases the bus and configures it as slave. At this point the ISSS-bit (SPIFLAG/B register) reflect the status of the $\overline{\text{SPIDS}}$ pin. 0 = Multi-master error detection disabled 1 = Multi-master error detection enabled Note that /SPIDS pin is the chip select for SPI in slave mode.
5	DMISO	**Disable MISO Pin.** Disables MISO as an output when a master wishes to transmit to various slaves at one time (broadcast). Only one slave is allowed to transmit data back to the master. Except for the slave from whom the master wishes to receive, all other slaves should have this bit set. 0 = MISO enabled 1 = MISO disabled
6	Reserved	
8–7	WL	**Word Length.** 00 = 8 bits 01 = 16 bits 10 = 32 bits
9	MSBF	**Most Significant Byte First.** 0 = LSB sent/received first 1 = MSB sent/received first
10	CPHASE	**Clock Phase.** Selects the transfer format. 0 = SPICLK starts toggling at the middle of 1st data bit 1 = SPICLK starts toggling at the start of 1st data bit
11	CLKPL	**Clock Polarity.** 0 = Active high SPICLK (SPICLK low is the idle state) 1 = Active low SPICLK (SPICLK high is the idle state)
12	SPIMS	**SPI Master Select.** Configures SPI module as master or slave. 0 = Device is a slave device 1 = Device is a master device

Table A-4. SPICTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
13	OPD	**Open Drain Output Enable.** Enables open drain data output enable (for MOSI and MISO). 0 = Normal 1 = Open-drain
14	SPIEN	**SPI Port Enable.** 0 = SPI module is disabled 1 = SPI module is enabled
15	PACKEN	**Packing Enable.** 0 = No packing 1 = 8 to16-bit packing Note: This bit may be 1 only when WL = 00 (8-bit transfer). When in transmit mode, the PACKEN bit unpacks data.
16	SGN	**Sign Extend.** 0 = No sign extension 1 = Sign extension
17	SMLS	**Seamless Transfer**. 0 = Seamless transfer disabled. After each word transfer there is a delay before the next word transfer starts. The delay is 2.5 SPICLK cycles 1 = Seamless transfer enabled. There is no delay before the next word starts, a seamless operation. This operation is not supported in mode TIMOD1-0 = 00 and CPHASE=0 for all modes.
18	TXFLSH	**Flush Transmit Buffer.** Write a 1 to this bit to clear TXSPI. 0 = TXSPI not cleared 1 = TXSPI cleared
19	RXFLSH	**Clear RXSPI.** Write a 1 to this bit to clear RXSPI. 0 = RXSPI not cleared 1 = RXSPI cleared
20	ILPBK	**Internal Loop Back.** This mode interconnects the MOSI to MISO pin. This mode only works in master mode. 0 = No internal loopback 1 = Internal loopback enabled
31–21	Reserved	

DMA Configuration Registers (SPIDMAC, SPIDMACB)

These 17-bit SPI registers are used to control DMA transfers and are shown in Figure A-6 and described in Table A-5.

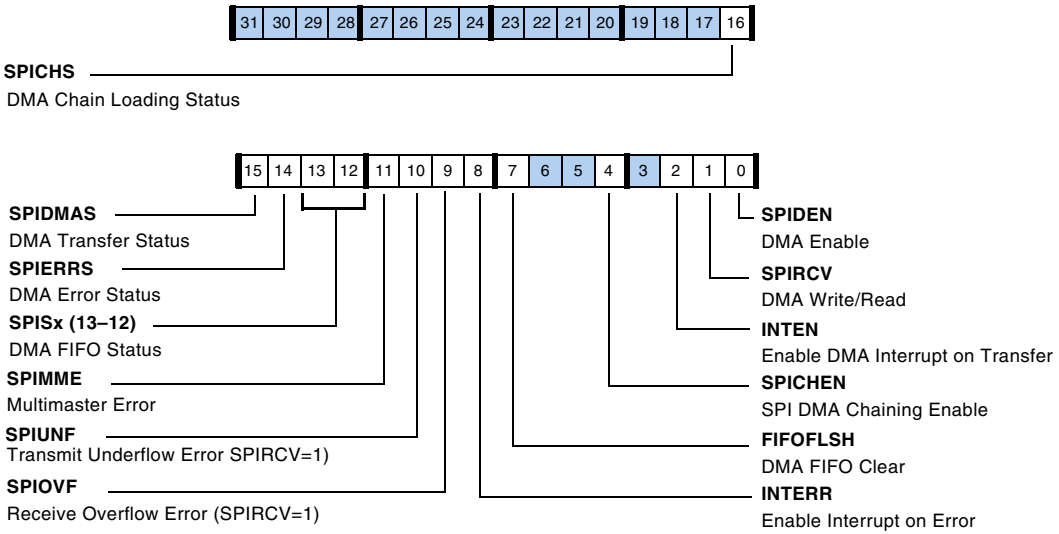


Figure A-6. SPIDMAC, SPIDMACB Registers

Table A-5. SPIDMAC, SPIDMACB Register Bit Descriptions

Bit	Name	Description
0	SPIDEN	**DMA Enable.** 0 = Disable 1 = Enable
1	SPIRCV	**DMA Write/Read.** 0 = SPI transmit (read from internal memory) 1 = SPI receive (write to internal memory)
2	INTEN	**Enable DMA Interrupt on Transfer.** 0 = Disable 1 = Enable
3	Reserved	

Table A-5. SPIDMAC, SPIDMACB Register Bit Descriptions (Cont'd)

Bit	Name	Description
4	SPICHEN	**SPI DMA Chaining Enable.** 0 = Disable 1 = Enable
6–5	Reserved	
7	FIFOFLSH	**DMA FIFO Clear.** 0 = Disable 1 = Enable
8	INTERR	**Enable Interrupt on Error.** 0 = Disable 1 = Enable
9	SPIOVF	**Receive OverFlow Error (SPIRCV = 1).** 0 = Successful transfer 1 = Error – data received with RXSPI full
10	SPIUNF	**Transmit Underflow Error (SPIRCV = 0).** 0 = Successful transfer 1 = Error occurred in transmission with no new data in TXSPI.
11	SPIMME	**Multimaster Error.** 0 = Successful transfer 1 = Error during transfer
13–12	SPISx	**DMA FIFO Status.** 00 = FIFO empty 01 = Reserved 10 = FIFO partially full 11 = FIFO full
14	SPIERRS	**DMA Error Status.** 0 = Successful DMA transfer 1 = Errors during DMA transfer
15	SPIDMAS	**DMA Transfer Status.** 0 = DMA idle 1 = DMA in progress

Table A-5. SPIDMAC, SPIDMACB Register Bit Descriptions (Cont'd)

Bit	Name	Description
16	SPICHS	**DMA Chain Loading Status.** 0 = Chain idle 1 = Chain loading in progress
31–17	Reserved	

SPI Baud Rate Registers (SPIBAUD, SPIBAUDB)

These SPI registers are 32-bit read/write registers that are used to set the bit transfer rate for a master device. When configured as slaves, the value written to these registers is ignored. The (SPIBAUDx) registers can be read from or written to at any time. Bit descriptions are provided in Table A-6.

Table A-6. SPIBAUD, SPIBAUDB Register Bit descriptions

Bit	Name	Description
0	Reserved	
15–1	BAUDR	**Baud Rate Enable.** Enables the SPICLK per the equation: SPICLK baud rate = peripheral clock (PCLK)/4 x BAUDR) Default=0
31–16	Reserved	

Note that this baud rate equation applies to master mode operation only. For slave mode operation, refer to the *ADSP-21362/3/4/5/6 SHARC Processor* data sheet.

SPI Port Status (SPISTAT, SPISTATB) Registers

The SPISTAT and SPISTATB registers are 32-bit read-only registers (bits 31–18 are reserved) used to detect when an SPI transfer is complete, if transmission/reception errors occur, and the status of the TXSPI and RXSPI FIFOs. The bit settings for these registers are shown in Figure A-7 and described in Table A-7.

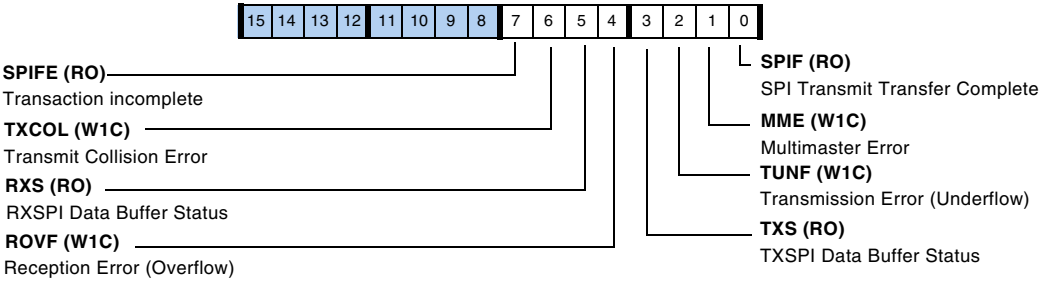


Figure A-7. SPISTAT, SPISTATB Registers

Table A-7. SPISTAT Register Bit Descriptions

Bit	Name	Description
0	SPIF	**SPI Transmit or Receive Transfer Complete.** SPIF is set when an SPI single-word transfer is complete.
1	MME	**Multimaster Error or Mode-fault Error.** MME is set in a master device when some other device tries to become the master.
2	TUNF	**Transmission Error.** TUNF is set when transmission occurred with no new data in the TXSPI register.
3	TXS	**Transmit Data Buffer Status.** TXSPI data buffer status. 0 = Empty 1 = Full
4	ROVF	**Reception Error.** ROVF is set when data is received with receive buffer full.
5	RXS	**Receive Data Buffer Status.** 0 = Empty 1 = Full
6	TXCOL	**Transmit Collision Error.** When TXCOL is set, it is possible that corrupt data was transmitted.
7	SPIFE	**External Transaction Complete.** Set (= 1) when the SPI transaction is complete on the external interface.
31–8	Reserved	

SPI Port Flags Registers (SPIFLG, SPIFLGB)

The SPIFLG and SPIFLGB registers are used to enable individual SPI slave-select lines when the SPI is enabled as a master. This 32-bit register (bits 31–12 are reserved) is ignored if the SPI is programmed as s slave. The bit settings for these registers are shown in Figure A-8 and described in Table A-8.

Note that the primary SPI can connect the slave select signals to the core FLAG3-0 pins, the parallel port, or the DAI pins. The secondary SPI (SPIB) slave selects can only be routed to the DAI pins.

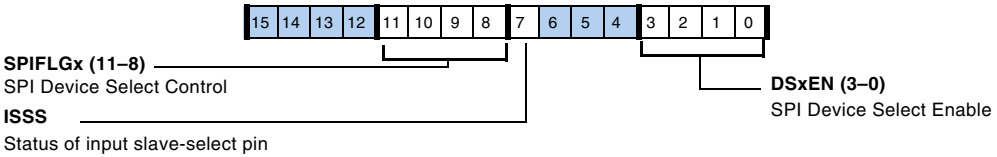

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SPIFLGx (11–8)
SPI Device Select Control
ISSS
Status of input slave-select pin
DSxEN (3–0)
SPI Device Select Enable

Figure A-8. SPIFLG, SPIFLGB Registers

Table A-8. SPIFLG, SPIFLGB Register Bit Descriptions

Bit	Name	Description
3–0	DSxEN	**SPI Device Select Output Enable.** Enables or disables the default Chip selects (SPI to FLAG3–0, SPIB to DAI pins) as outputs to be used for SPI slave-select. For CPHASE=0, the chip select is automatically controlled by assertion of chip select only during transfers. 0 = Disable chip select 3–0 as output 1 = Enable chip select 3–0 as output Note: if slave selects for the parallel port are required, see "Parallel Port Pin Multiplexing" on page 14-20.
6–4	Reserved	
7	ISSS	**Input Service Slave Select.** This bit reflects the status of the $\overline{\text{SPIDS}}$ pin in a multi-master system. Note that the ISSEN bit in the SPICTL register must be set to enable the status logic.

Table A-8. SPIFLG, SPIFLGB Register Bit Descriptions (Cont'd)

Bit	Name	Description
11–8	SPIFLGx	**SPI Chip Select Level Control.** For CPHASE=1 the SPI does not control the chip selects. These bits give SW control to toggle the individual chip selects. 0 = Chip Select High Level 1 = Chip Select Low Level Note: the status of the FLAG3–0 pins cannot be polled from the FLAGS register using the SPI.
31–12	Reserved	

RXSPI Shadow Registers (RXSPI_SHADOW, RXSPIB_SHADOW)

These registers act as shadow registers for the receive data buffer, RXSPI and RXSPIB registers, and are used to aid software debugging. Although these registers reside at a different addresses from the RXSPI and RXSPIB registers, their contents are identical. When a software read of RXSPIx occurs, the RXS bit is cleared and an SPI transfer may be initiated (if TIMOD=00). No such hardware action occurs when the shadow register is read.

Pulse Width Modulation Registers

The following registers control the operation of pulse width modulation on the ADSP-2136x processor processor.

PWM Global Control Register (PWMGCTL)

This register enables or disables the four PWM groups, in any combination and provides synchronization acrossthe groups. Note that disable bits have higher priority over the enable bits (bit 1 higher as bit 0 and so on). This 16-bit read/write register is shown in Figure A-9.

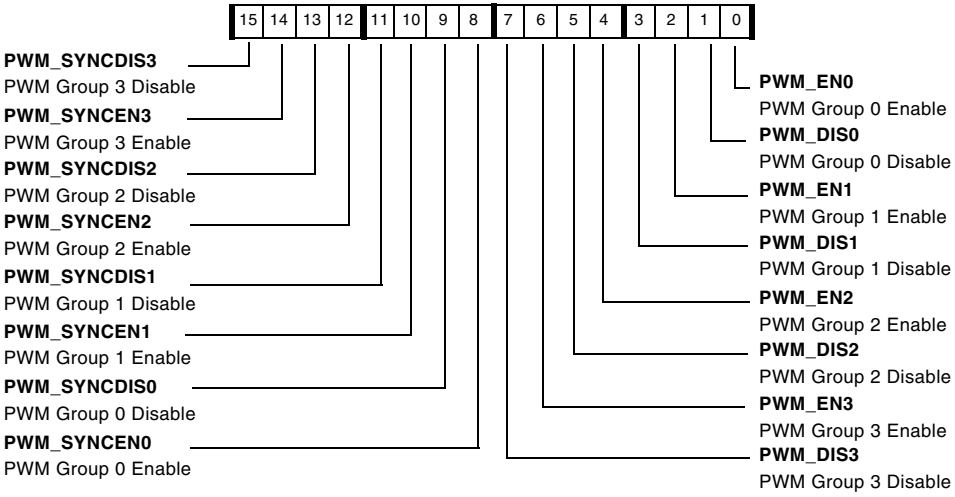


Figure A-9. PWMGCTL Register

PWM Global Status Register (PWMGSTAT)

This register provides the status of each PWM group. The bits in this register (Figure A-11, Table A-9) are W1C-type bits (write one-to-clear).

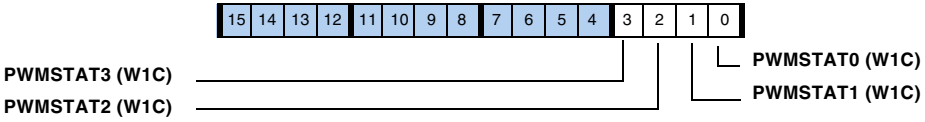


Figure A-10. PWMGSTAT Register

Table A-9. PWMGSTAT Register Bit Descriptions

Bit	Name	Function
0	PWM_STAT0	PWM group 0 period completion status
1	PWM_STAT1	PWM group 1 period completion status
2	PWM_STAT2	PWM group 2 period completion status

Table A-9. PWMGSTAT Register Bit Descriptions (Cont'd)

Bit	Name	Function
3	PWM_STAT3	PWM group 3 period completion status
15–4	Reserved	

PWM Control Register (PWMCTLx)

These registers, shown in Figure A-11 and described in Table A-10, are used to set the operating modes of each PWM block. They also allow programs to disable interrupts from individual groups.

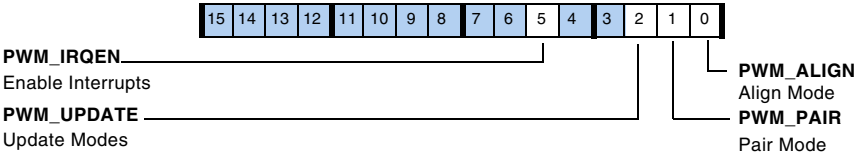


Figure A-11. PWMCTLx Register

Table A-10. PWMCTLx Register Bit Descriptions

Bit	Name	Description
0	PWM_ALIGN	**Align Mode.** 0 = Edge-aligned. The PWM waveform is left-justified in the period window. 1 = Center-aligned. The PWM waveform is symmetrical.
1	PWM_PAIR	**Pair Mode.** 0 = Non-paired mode. The PWM generates independent signals 1 = Paired mode. The PWM generates complementary signals on two outputs.
2	PWM_UPDATE	**Update Mode.** 0 = Single update mode. The duty cycle values are programmable only once per PWM period. The resulting PWM patterns are symmetrical about the mid-point of the PWM period. 1 = Double update mode. A second update of the PWM registers is implemented at the mid-point of the PWM period.

Table A-10. PWMCTLx Register Bit Descriptions (Cont'd)

Bit	Name	Description
4–3	Reserved	
5	PWM_IRQEN	**Enable PWM Interrupts.** Enables interrupts. 0 = Interrupts not enabled 1 = Interrupts enabled
15–6	Reserved	

PWM Status Registers (PWMSTATx)

These 16-bit, read-only registers, shown in Figure A-12 and described in Table A-11, report the status of the phase and mode for each PWM group.

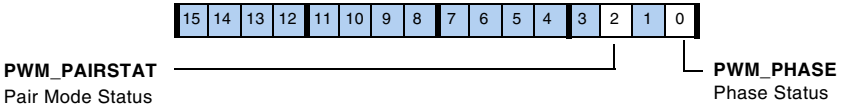


Figure A-12. PWMSTATx Register

Table A-11. PWMSTATx Register Bit Descriptions

Bit	Name	Description
0	PWM_PHASE	**PWM Phase Status.** Set during operation in the second half of each PWM period. Allows programs to determine the particular half-cycle (first or second) during implementation of the PWM-SYNC interrupt service routine, if required. 0 = First half 1 = Second half
1	Reserved	
2	PWM_PAIRSTAT	**PWM Paired Mode Status.** 0 = Inactive paired mode 1 = Active paired mode
15–3	Reserved	

PWM Period Registers (PWMPERIODx)

These 16-bit, read/write registers control the unsigned period of the four PWM groups.

PWM Output Disable Registers (PWMSEGx)

These 16-bit read/write registers, shown in Figure A-13 and described in Table A-12, control the output signals of the four PWM groups.

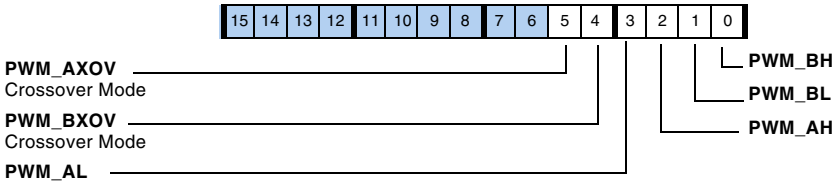


Figure A-13. PWMSEGx Register

Table A-12. PWMSEGx Register Bit Descriptions

Bit	Name	Description
0	PWM_BH	**Channel B High Disable.** Enables or disables the channel B output signal. 0 = Enable 1 = Disable
1	PWM_BL	**Channel B Low Disable.** Enables or disables the channel B output signal. 0 = Enable 1 = Disable
2	PWM_AH	**Channel A High Disable.** Enables or disables the channel A output signal. 0 = Enable 1 = Disable
3	PWM_AL	**Channel A Low Disable.** Enables or disables the channel A output signal. 0 = Enable 1 = Disable

Table A-12. PWMSEGx Register Bit Descriptions (Cont'd)

Bit	Name	Description
4	PWM_BXOV	**B Signal Pair Crossover Enable.** Enable cross over mode for the PWM_BH and PWM_BL signal pair. 0 = Disable 1 = Enable
5	PWM_AXOV	**A Signal Pair Crossover Enable.** Enable cross over mode for the PWM_AH and PWM_AL signal pair. 0 = Disable 1 = Enable
15–6	Reserved	

PWM Polarity Select Registers (PWMPOLx)

These 16-bit registers, shown in Figure A-14 and described in Table A-13, control the polarity of the four PWM groups which can be set to either active high or active low. Note that bit 1 has priority over bit 0, bit 3 over bit 2 and so on.

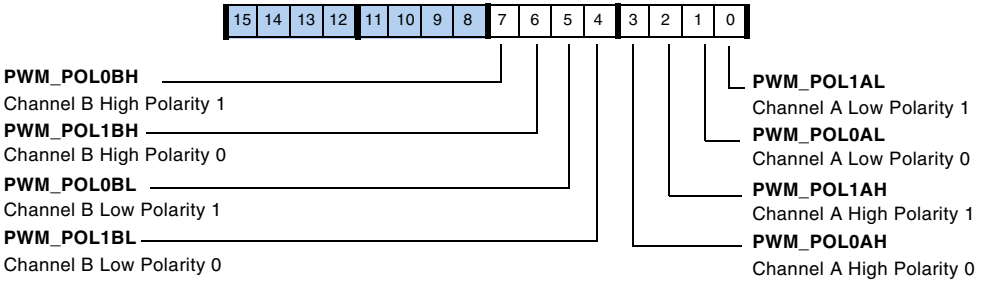


Figure A-14. PWMPOLx Register

Table A-13. PWMPOLx Register Bit Descriptions

Bit	Name	Description
0	PWM_POL1AL	Write 1 to set channel A low polarity 1
1	PWM_POL0AL	Write to set channel A low polarity 0

Table A-13. PWMPOLx Register Bit Descriptions (Cont'd)

Bit	Name	Description
2	PWM_POL1AH	Write 1 to set channel A high polarity 1
3	PWM_POL0AH	Write 1 to set channel A high polarity 0
4	PWM_POL1BL	Write 1 to set channel B low polarity 1
5	PWM_POL0BL	Write 1 to set channel B low polarity 0
6	PWM_POL1BH	Write 1 to set channel B high polarity 1
7	PWM_POL0BH	Write 1 to set channel B high polarity 0
15–8	Reserved	

PWM Channel Duty Control Registers (PWMAx, PWMBx)

The 16-bit duty-cycle control registers directly control the A/B (two's complement) duty cycles of the two pairs of PWM signals.

PWM Channel Low Duty Control Registers (PWMALx, PWMBLx)

The 16-bit duty-cycle control registers directly control the AL/BL duty cycles (two's complement) of the non-paired PWM signals. These can be different from the AH/BH cycles.

PWM Dead Time Registers (PWMDTx)

These 16-bit registers set up a short time delay (10-bit, unsigned) between turning off one PWM signal and turning on its complementary signal.

PWM Debug Status Registers (PWMDBGx)

These 16-bit read-only registers aid in software debug activities.

Table A-14. PWMDBGx Register Bit Descriptions

Bit	Name	Function
0	PWM_AL	Channel A low output signal for S/W observation
1	PWM_AH	Channel A high output signal for S/W observation
2	PWM_BL	Channel B low output signal for S/W observation
3	PWM_BH	Channel B high output signal for S/W observation
15–4	Reserved	

Peripherals Routed Through the DAI

The following sections provide information on the peripherals that are explicitly routed through the digital applications interface. For more information, see “DAI Signal Routing Unit Registers” on page A-80.

Serial Port Registers

The following section describes serial port (SPORT) registers.

SPORT Serial Control Registers (SPCTLx)

The `SPCTLx` registers are transmit and receive control registers for the corresponding serial ports (SPORT 0 through 5). Bit descriptions are provided in Table A-15.

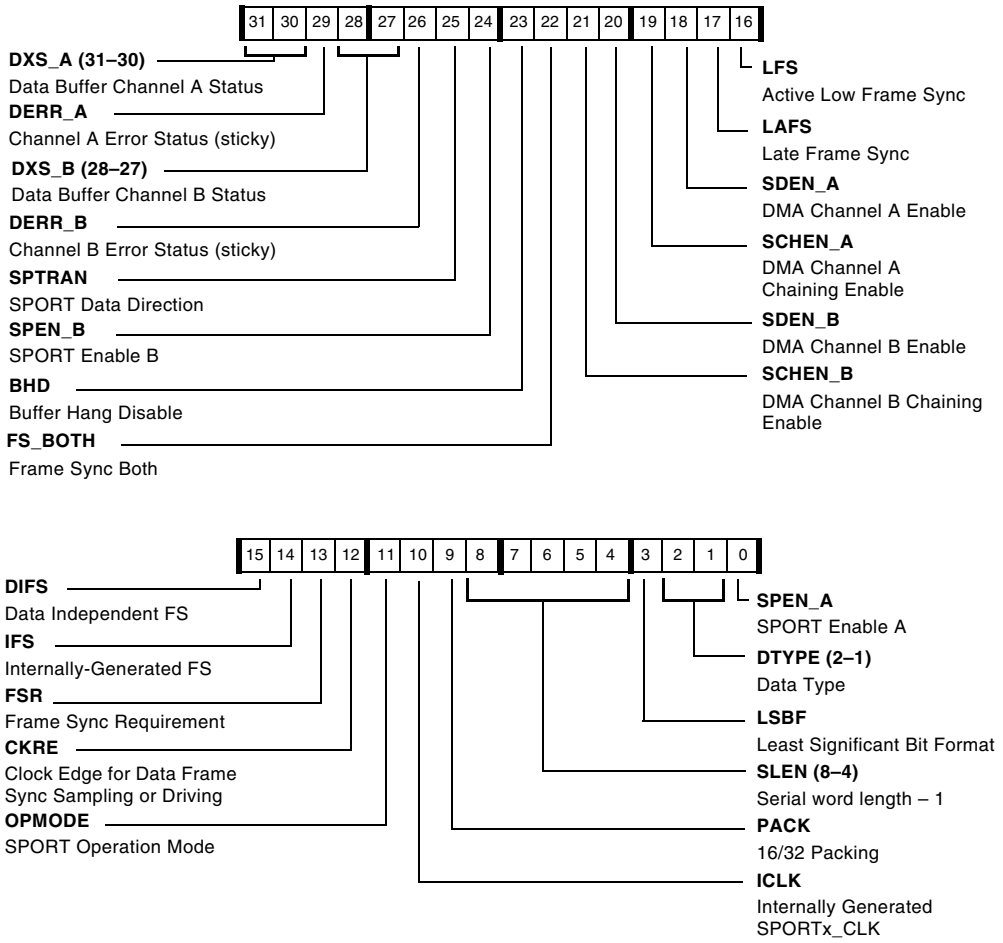


Figure A-15. SPCTLx Register for Standard Serial Mode

Table A-15. SPCTLx Register Bit Descriptions (Standard Serial)

Bit	Name	Description
0	SPEN_A	**Enable Channel A Serial Port.** 0 = Serial port A channel disabled 1 = Serial port A channel enabled
2–1	DTYPE	**Data Type Select.** Selects the data type formatting for standard serial mode transmissions. For Standard serial mode, selection of companding mode and MSB format are exclusive: **Serial Data Type Formatting** 00 Right-justify, zero-fill unused MSBs 01 Right-justify, sign-extend unused MSBs 10 Compand using μ-law 11 Compand using A-law The transmit shift register does not zero-fill or sign-extend transmit data words; this only takes place for the receive shift register.
3	LSBF	**Serial Word Endian Select.** 0 = Big endian (MSB first) 1 = Little endian (LSB first)
8–4	SLEN	**Serial Word Length Select.** Selects the word length in bits. For DSP serial and multichannel modes, word sizes can be from 3 bits (SLEN = 2) to 32 bits (SLEN = 31).
9	PACK	**16-bit to 32-bit Word Packing Enable.** 0 = Disable 16- to 32-bit word packing 1 = Enable 16- to 32-bit word packing
10	ICLK	**Internal Clock Select.** 0 = Select external clock 1 = Select internal clock
11	OPMODE	**Sport Operation Mode.** 0 = DSP serial/multichannel mode if cleared
12	CKRE	**Clock Rising Edge Select.** Determines clock signal to sample data and the frame sync selection. 0 = Falling edge 1 = Rising edge
13	FSR	**Frame Sync Required Select.** Selects whether the serial port requires (if set, = 1) or does not require a transfer frame sync (if cleared, = 0).

Table A-15. SPCTLx Register Bit Descriptions (Standard Serial) (Cont'd)

Bit	Name	Description
14	IFS	**Internal Frame Sync Select.** Selects whether the serial port uses an internally generated frame sync (if set, = 1) or uses an external frame sync (if cleared, = 0).
15	DIFS	**Data Independent Frame Sync Select.** 1 = Serial port uses a data-independent frame sync (sync at selected interval) 0 = Serial port uses a data-dependent frame sync (sync when TX FIFO is not empty or when RX FIFO is not full).
16	LFS	**Active Low Frame Sync Select.** This bit selects the logic level of the (transmit or receive) frame sync signals. This bit selects an active low frame sync (if set, = 1) or active high frame sync (if cleared, = 0).
17	LAFS	**Late Transmit Frame Sync Select.** This bit selects when to generate the frame sync signal. This bit selects a late frame sync if set (= 1) during the first bit of each data word. This bit selects an early frame sync if cleared (= 0) during the serial clock cycle immediately preceding the first data bit 0 = Early frame sync (FS before first bit) 1 = Late frame sync (FS during first bit)
18	SDEN_A	**Enable Channel A Serial Port DMA.** 0 = Disable serial port channel A DMA 1 = Enable serial port channel A DMA
19	SCHEN_A	**Enable Channel A Serial Port DMA Chaining.** 0 = Disable serial port channel A DMA chaining 1 = Enable serial port channel A DMA chaining
20	SDEN_B	**Enable Channel B Serial Port DMA.** 0 = Disable serial port channel B DMA 1 = Enable serial port channel B DMA
21	SCHEN_B	**Enable Channel B Serial Port DMA Chaining.** 0 = Disable serial port channel B DMA chaining 1 = Enable serial port channel B DMA chaining

Table A-15. SPCTLx Register Bit Descriptions (Standard Serial) (Cont'd)

Bit	Name	Description
22	FS_BOTH	**FS Both Enable.** This bit applies when the SPORTS channels A and B are configured to transmit/receive data. If set (= 1), this bit issues frame sync only when data is present in both transmit buffers, TXSPxA and TXSPxB. If cleared (= 0), a frame sync is issued if data is present in either transmit buffers. When a SPORT is configured as a receiver, if FS_BOTH is set (= 1), frame sync is issued only when both the RX FIFOs (RXSPxA and RXSPxB) are not full. If only channel A or channel B is selected, the frame sync behaves as if FS_BOTH is cleared (= 0). If both A and B channels are selected, the word select acts as if FS_BOTH is set (= 1). 0 = Issue FS if data is present in either transmit buffer. 1 = Issue FS if data is present in both transmit buffers.
23	BHD	**Buffer Hang Disable.** 0 = Indicates a core stall. The core stalls when it tries to write to a full transmit buffer or read an empty receive buffer FIFO. 1 = Ignore a core stall
24	SPEN_B	**Enable Channel B Serial Port.** 0 = Serial port A channel disabled 1 = Serial port A channel enabled
25	SPTRAN	**Data Direction Control.** This bit controls the data direction of the serial port channel A and B signals. When cleared (= 0) the SPORT is configured to receive on both channels A and B. When configured to receive, the RXSPxA and RXSPxB buffers are activated, while the receive shift registers are controlled by SPORTx_CLK and SPORTx_FS. The TXSPxA and TXSPxB buffers are inactive. When set (= 1) the SPORT is configured to transmit on both channels A and B. When configured to transmit, the TXSPxA and TXSPxB buffers are activated, while the transmit shift registers are controlled by SPORTx_CLK and SPORTx_FS. The RXSPxA and RXSPxB buffers are inactive.

Table A-15. SPCTLx Register Bit Descriptions (Standard Serial) (Cont'd)

Bit	Name	Description
26	DERR_B	**Channel B Error Status (sticky, read-only).** Indicates if the serial transmit operation has underflowed or a receive operation has overflowed in the channel B data buffer. The error status bit (DERR_B) are set when the SPORTx_FS signal occurs from either an external or internal source while the TXSPxB buffer is empty. The internally-generated SPORTx_FS signal may be suppressed whenever TXSPxB is empty by clearing the DIFS control bit when SPTRAN = 1.
28–27	DXS_B	**Channel B Data Buffer Status (read-only).** Indicates the status of the serial port's channel B data buffer (RXSPxB or TXSPxB) as follows: 00 = empty, 10 = partially full, 11 = full
29	DERR_A	**Channel A Error Status (sticky, read-only).** Refer to DERR_B
31–30	DXS_A	**Channel A Data Buffer Status (read-only).** Refer to DXS_B

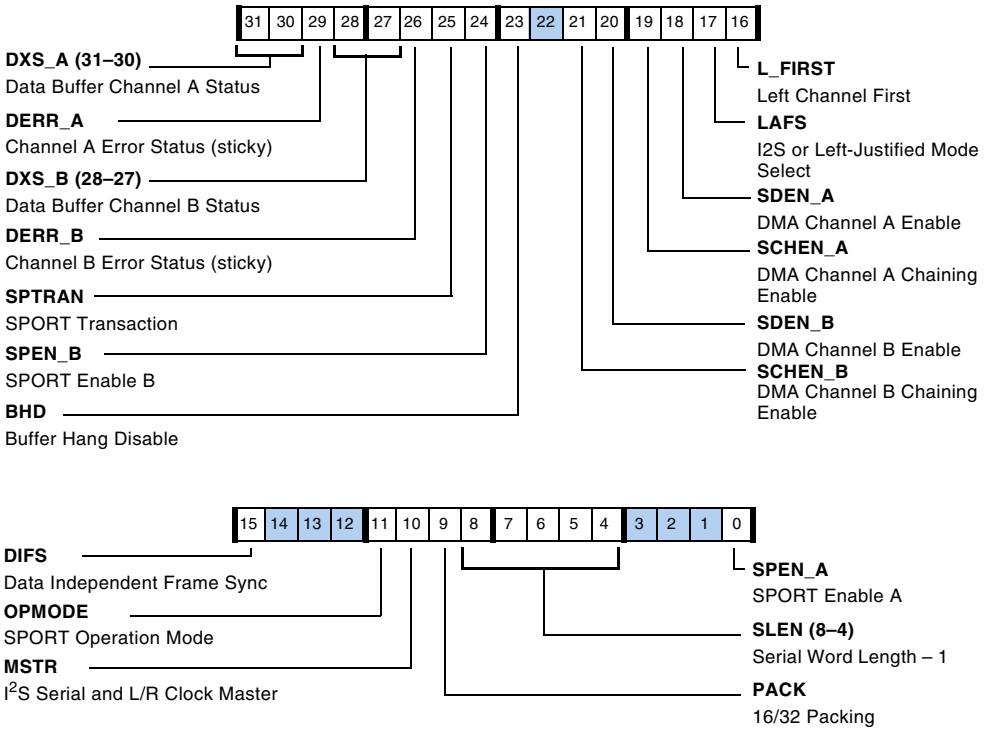


Figure A-16. SPCTLx Register for I^{2}S and Left-Justified Modes

Table A-16. SPCTLx Register Bit Descriptions (I^{2}S, Left-Justified)

Bit	Name	Description
0	SPEN_A	**Enable Channel A Serial Port.** 0 = Serial port A channel disabled 1 = Serial port A channel enabled
3–1	Reserved	
8–4	SLEN	**Serial Word Length Select.** Selects the word length in bits. Word sizes can be from 8 bits (SLEN = 7) to 32 bits (SLEN = 31).

Table A-16. SPCTLx Register Bit Descriptions (I^2S, Left-Justified) (Cont'd)

Bit	Name	Description
9	PACK	**16-bit to 32-bit Word Packing Enable.** 0 = Disable 16- to 32-bit word packing 1 = Enable 16- to 32-bit word packing
10	MSTR	**Master Clock Select.** 0 = Select external clock and frame sync 1 = Select internal clock and frame sync
11	OPMODE	**Sport Operation Mode.** 1 = Selects the I^2S or left-justified mode Bit 17 is used to select either of both modes
14–12	Reserved	
15	DIFS	**Data Independent Frame Sync Select.** 0 = Serial port uses a data-dependent frame sync (sync when TX FIFO is not empty or when RX FIFO is not full). 1 = Serial port uses a data-independent frame sync (sync at selected interval)
16	L_FIRST	**Left Channel Word First Select.** Selects left or right channel Word first. To select the channel order, set the L_FIRST bit (= 1) to transmit or receive on left channel first, or clear the L_FIRST bit (= 0) to transmit or receive on right channel first.
17	LAFS	**I2S or Left-Justified Mode Select.** 0 = I2S mode 1 = Left-justified mode See also bit 11 of this register.
18	SDEN_A	**Enable Channel A Serial Port DMA.** 0 = Disable serial port channel A DMA 1 = Enable serial port channel A DMA
19	SCHEN_A	**Enable Channel A Serial Port DMA Chaining.** 0 = Disable serial port channel A DMA chaining 1 = Enable serial port channel A DMA chaining
20	SDEN_B	**Enable Channel B Serial Port DMA.** 0 = Disable serial port channel B DMA 1 = Enable serial port channel B DMA

Table A-16. SPCTLx Register Bit Descriptions (I^2S, Left-Justified) (Cont'd)

Bit	Name	Description
21	SCHEN_B	**Enable Channel B Serial Port DMA Chaining.** 0 = Disable serial port channel B DMA chaining 1 = Enable serial port channel B DMA chaining
22	Reserved	
23	BHD	**Buffer Hang Disable.** 0 = Indicates a core stall. The core stalls when it tries to write to a full transmit buffer or read an empty receive buffer FIFO. 1 = Ignore a core hang
24	SPEN_B	**Enable Channel B Serial Port.** 0 = Serial port A channel disabled 1 = Serial port A channel enabled
25	SPTRAN	**Data Direction Control.** This bit controls the data direction of the serial port channel A and B signals. 0 = SPORT is configured to receive on both channels A and B. When configured to receive, the RXSPxA and RXSPxB buffers are activated, while the receive shift registers are controlled by SPORTx_CLK and SPORTx_FS. The TXSPxA and TXSPxB buffers are inactive. 1 = SPORT is configured to transmit on both channels A and B. When configured to transmit, the TXSPxA and TXSPxB buffers are activated, while the transmit shift registers are controlled by SPORTx_CLK and SPORTx_FS. The RXSPxA and RXSPxB buffers are inactive.
26	DERR_B	**Channel B Error Status (sticky, read-only).** Indicates if the serial transmit operation has underflowed or a receive operation has overflowed in the channel B data buffer. The error status bit (DERR_B) are set when the SPORTx_FS signal occurs from either an external or internal source while the TXSPxB buffer is empty. The internally-generated SPORTx_FS signal may be suppressed whenever TXSPxB is empty by clearing the DIFS control bit when SPTRAN = 1.
28–27	DXS_B	**Channel B Data Buffer Status (read-only).** Indicates the status of the serial port's channel B data buffer (RXSPxB or TXSPxB) as follows: 00 = Empty 10 = Partially full 11 = Full

Table A-16. SPCTLx Register Bit Descriptions (I^2S, Left-Justified) (Cont'd)

Bit	Name	Description
29	DERR_A	**Channel A Error Status (sticky, read-only).** Refer to DERR_B
31–30	DXS_A	**Channel A Data Buffer Status (read-only).** Refer to DXS_B

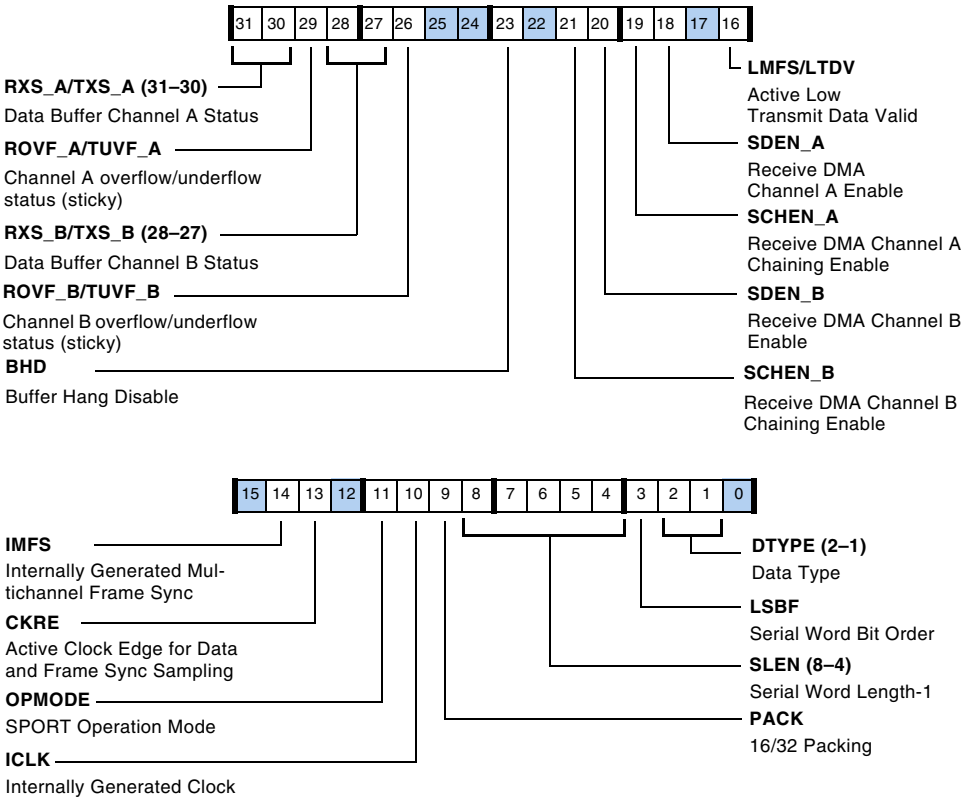


Figure A-17. SPCTLx Register – Multichannel Mode

Table A-17. SPCTLx Register Bit Descriptions (Multichannel)

Bit	Name	Description
0	Reserved	
2–1	DTYPE	**Data Type Select.** Selects the data type formatting for standard serial mode transmissions. For Standard serial mode, selection of companding mode and MSB format are exclusive: **Multichannel** **Data Type Formatting** 00 Right-justify, zero-fill unused MSBs 01 Right-justify, sign-extend unused MSBs 10 Compand using μ-law 11 Compand using A-law The transmit shift register does not zero-fill or sign-extend transmit data words; this only takes place for the receive shift register.
3	LSBF	**Serial Word Endian Select.** 0 = Big endian (MSB first) 1 = Little endian (LSB first)
8–4	SLEN	**Serial Word Length Select.** Selects the word length in bits. Word sizes can be from 3 bits (SLEN = 2) to 32 bits (SLEN = 31).
9	PACK	**16-bit to 32-bit Word Packing Enable.** 0 = Disable 16- to 32-bit word packing 1 = Enable 16- to 32-bit word packing
10	ICLK	**Internal Clock Select.** Select the receiver clock for SPORT(1/3/5) 0 = Select external receiver clock 1 = Select internal receiver clock Note: For the transmitter clock (SPORT0/2/4) bit is reserved
11	OPMODE	**Sport Operation Mode.** 0 = multichannel mode Note for multichannel operation, the SPMCTLxy registers must be programmed
12	CKRE	**Clock Rising Edge Select.** Determines clock signal to sample data and the frame sync selection. 0 = Falling edge 1 = Rising edge
13	Reserved	

Table A-17. SPCTLx Register Bit Descriptions (Multichannel) (Cont'd)

Bit	Name	Description
14	IMFS	**Internal Multichannel Frame Sync Select.** Selects whether the serial port uses an internally generated frame sync (if set, = 1) or uses an external frame sync (if cleared, = 0). This bit only valid for the receiver (SPORT1/3/5). Note: For the transmitter (SPORT0/2/4) this bit is reserved.
15	Reserved	
16	LMFS LTDV	**Active Low Multichannel Frame Sync Select.** Selects an active high or low frame sync for the Sport receiver (SPORT1/3/5). **Active Low Transmit Data Valid Select.** Selects an active high or low frame sync for the Sport transmitter (SPORT0/2/4).
17	Reserved	
18	SDEN_A	**Enable Channel A Serial Port DMA.** 0 = Disable serial port channel A DMA 1 = Enable serial port channel A DMA
19	SCHEN_A	**Enable Channel A Serial Port DMA Chaining.** 0 = Disable serial port channel A DMA chaining 1 = Enable serial port channel A DMA chaining
20	SDEN_B	**Enable Channel B Serial Port DMA.** 0 = Disable serial port channel B DMA 1 = Enable serial port channel B DMA
21	SCHEN_B	**Enable Channel B Serial Port DMA Chaining.** 0 = Disable serial port channel B DMA chaining 1 = Enable serial port channel B DMA chaining
22	Reserved	
23	BHD	**Buffer Hang Disable.** 0 = Indicates a core stall. The core stalls when it tries to write to a full transmit buffer or read an empty receive buffer FIFO. 1 = Ignore a core hang
25–24	Reserved	

Table A-17. SPCTLx Register Bit Descriptions (Multichannel) (Cont'd)

Bit	Name	Description
26	ROVF_B	**Channel B Error Status (sticky, read-only).** Indicates if the serial receive operation has overflowed (SPORT1/3/5) in the channel B data buffer.
	TUVF_B	**Channel B Error Status (sticky, read-only).** Indicates if the serial transmit operation has underflowed (SPORT0/2/4) in the channel B data buffer.
28–27	RXS_B	**Channel B Data Buffer Status (read-only).** Indicates the status of the serial port's receive channel B (SPORT1/3/5) data buffer as follows: 00 = empty, 10 = partially full, 11 = full
	TXS_B	**Channel B Data Buffer Status (read-only).** Indicates the status of the serial port's transmit channel B (SPORT0/2/4) data buffer as follows: 00 = empty, 10 = partially full, 11 = full
29	ROVF_A	**Channel A Error Status (sticky, read-only).** Indicates if the serial receive operation has overflowed (SPORT1/3/5) in the channel B data buffer.
	TUVF_A	**Channel A Error Status (sticky, read-only).** Indicates if the serial transmit operation has underflowed (SPORT0/2/4) in the channel B data buffer.
31–30	RXS_A	**Channel A Data Buffer Status (read-only).** Indicates the status of the serial port's receive channel B (SPORT1/3/5) data buffer as follows: 00 = empty, 10 = partially full, 11 = full
	TXS_A	**Channel A Data Buffer Status (read-only).** Indicates the status of the serial port's transmit channel B (SPORT0/2/4) data buffer as follows: 00 = empty, 10 = partially full, 11 = full

SPORT Multichannel Control Registers (SPMCTLxy)

In TDM mode, the SPORTs are working in defined pairs (01/23/45). The `SPMCTLxy` register is the multichannel control register for SPORTs including the standard and chained DMA status. (x = 0, 2, and 4; y = 1, 3, and 5, shown in Figure A-18).

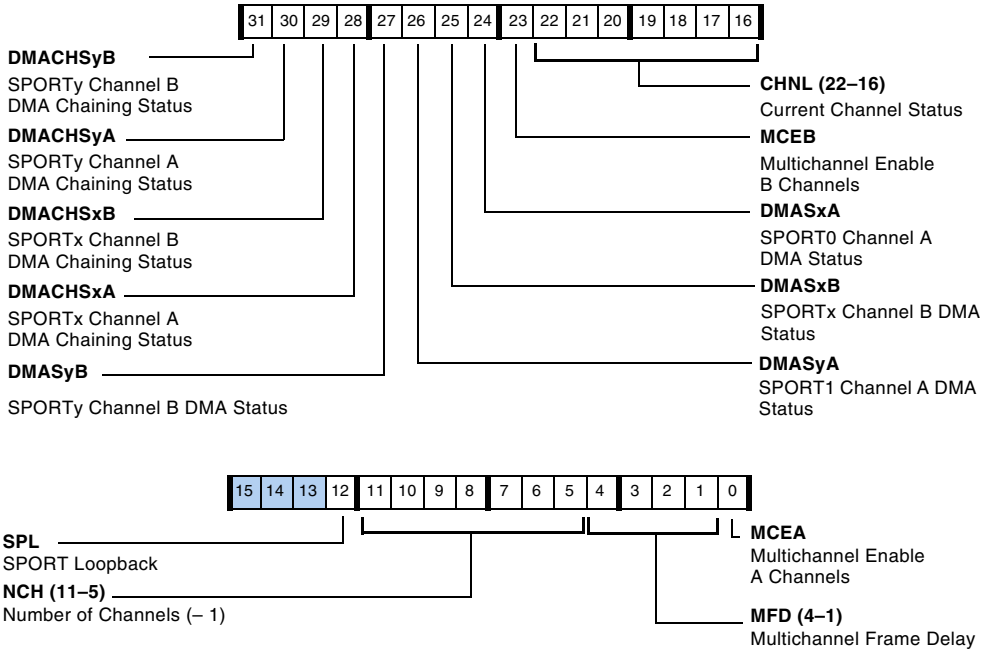


Figure A-18. SPMCTLxy Register – Multichannel Mode

Table A-18. SPMCTLxy Register Bit Descriptions

Bit	Name	Description
0	MCEA	**Multichannel Mode Enable.** Standard and multichannel modes only. One of two configuration bits that enable and disable multichannel mode on serial port channels. See also, OPMODE on page A-26. 0 = Disable multichannel operation 1 = Enable multichannel operation if OPMODE = 0
4–1	MFD	**Multichannel Frame Delay.** Sets the interval, in number of serial clock cycles, between the multichannel frame sync pulse and the first data bit. These bits provide support for different types of T1 interface devices. Valid values range from 0 to 15 with bits SPMCTL01[4:1], SPMCTL23[4:1], or SPMCTL45[4:1]. Values of 1 to15 correspond to the number of intervening serial clock cycles. A value of 0 corresponds to no delay. The multichannel frame sync pulse is concurrent with first data bit.
11–5	NCH	**Number of Multichannel Slots (minus one).** Selects the number of channel slots (maximum of 128) to use for multichannel operation. Valid values for actual number of channel slots range from 1 to 128. Use this formula to calculate the value for NCH: NCH = Actual number of channel slots – 1.
12	SPL	**SPORT Loopback Mode**. Enables if set (= 1) or disables if cleared (= 0) the channel loopback mode. Loopback mode enables developers to run internal tests and to debug applications. Loopback only works under the following SPORT configurations: • SPORT0 (configured to receive or transmit) together with SPORT1 (configured to transmit or receive). SPORT0 can only be paired with SPORT1, controlled via the SPL bit in the SPMCTL01 register. • SPORT2 (as a receiver or transmitter) together with SPORT3 (as a transmitter or receiver). SPORT2 can only be paired with SPORT3, controlled via the SPL bit in the SPMCTL23 register. • SPORT4 (configured to receive or transmit) together with SPORT5 (configured to transmit or receive). SPORT4 can only be paired with SPORT5, controlled via the SPL bit in the SPMCTL45 register. Either of the two paired SPORTs can be set up to transmit or receive, depending on their SPTRAN bit configurations.
15–13	Reserved	

Table A-18. SPMCTLxy Register Bit Descriptions (Cont'd)

Bit	Name	Description
22–16	CHNL	**Current Channel Selected.** Identifies the currently selected transmit channel slot (0 to 127). (Read-only, sticky)
23	MCEB	**Multichannel Enable, B Channels.** 0 = Disable 1 = Enable
27–24	DMASxy	**DMA Status.** Defines the DMA A/B channel status for the SPORTs x=0, 2, 4 and y=1, 3, 5 0 = DMA Inactive 1 = DMA Active (Read-only)
31–28	DMACHSxy	**DMA Chaining Status.** Defines the DMA Chaining A/B channel status for the SPORTs x=0, 2, 4 and y=1, 3, 5 0 = DMA Chain Loading inactive 1 = DMA Chain Loading active (Read-only)

SPORT Transmit Select Registers (MTxCSy)

Each bit, 31–0, set (= 1) in one of four `MTxCSy` registers corresponds to an active transmit channel, 127–0, on a multichannel mode serial port. When the `MTxCSy` registers activate a channel, the serial port transmits the word in that channel's position of the data stream. When a channel's bit in the `MTxCSy` register is cleared (= 0), the serial port's data transmit pin three-states during the channel's transmit time slot.

SPORT Transmit Compand Registers (MTxCCSy)

Each bit, 31–0, set (= 1) in one of four `MTxCCSx` registers corresponds to a companded transmit channel, 127–0, on a multichannel mode serial port. When the `MTCCSx` register activates companding for a channel, the serial port applies the companding from the `DTYPE` selection to the transmitted word in that channel's position of the data stream. When a channel's bit

in the MTCCSx register is cleared (= 0), the serial port does not compand the output during the channel's receive time slot.

SPORT Receive Select Registers (MRxCSx)

Each bit, 31–0, set (= 1) in one of the four MRCSx registers corresponds to an active receive channel, 127–0, on a multichannel mode serial port. When the MRxCSx register activates a channel, the serial port receives the word in that channel's position of the data stream and loads the word into the RXSPx buffer. When a channel's bit in the MRxCSx register is cleared (= 0), the serial port ignores any input during the channel's receive time slot.

SPORT Receive Compand Registers (MRxCCSx)

Each bit, 31–0, set (= 1) in the MRxCCSy registers corresponds to a companded receive channel, 127–0, on a multichannel mode serial port. When one of the four MRxCCSy registers activate companding for a channel, the serial port applies the companding from the DTYPE selection to the received word in that channel's position of the data stream. When a channel's bit in the MRxCCSy registers are cleared (= 0), the serial port does not compand the input during the channel's receive time slot.

SPORT Divisor Registers (DIVx)

This register, shown in Figure A-19 allows programs to set the frame sync divisor and clock divisor.

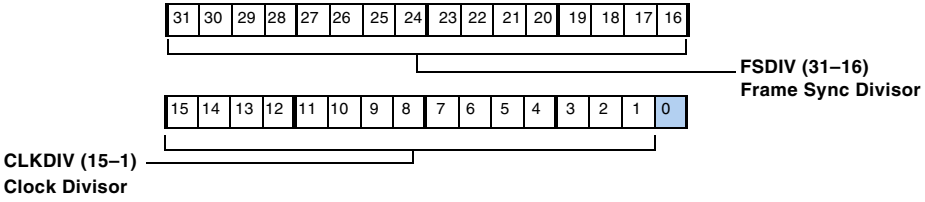


Figure A-19. DIVx Register

Input Data Port Registers

The input data port (IDP) provides an additional input path to the processor core. The IDP can be configured as 8 channels of serial data or 7 channels of serial data and a single channel of up to a 20-bit wide parallel data.

Input Data Port DMA Control Registers

For information on these registers, see "Standard DMA Parameter Registers" on page 2-6.

Input Data Port Control Register 0 (IDP_CTL0)

Use this register to configure and enable the IDP and each of its channels. The register is shown in Figure A-20 and described in Table A-19.

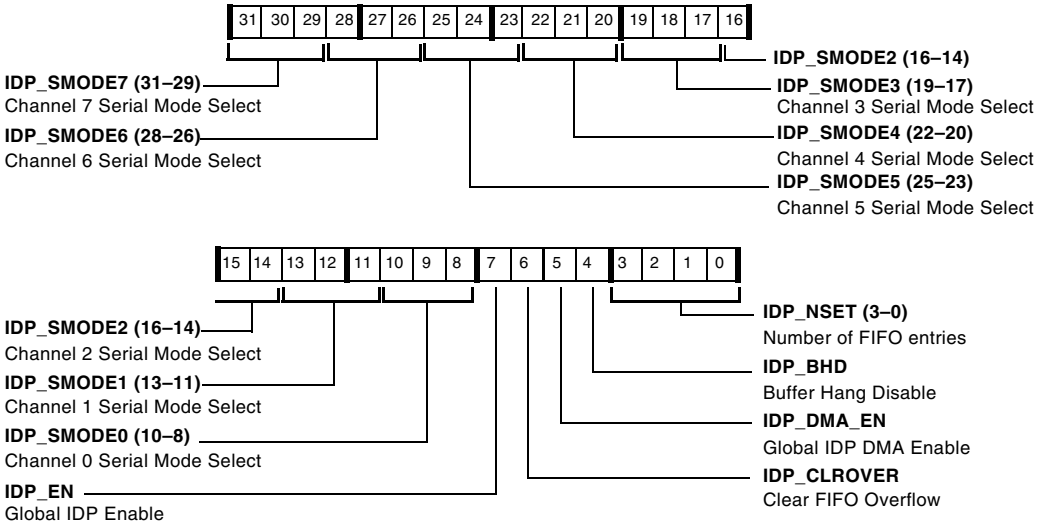


Figure A-20. IDP_CTL0 Register

Table A-19. IDP_CTL0 Register Bit Descriptions

Bits	Name	Description
3–0	IDP_NSET	**Threshold Interrupt.** The contents of the IDP_NSET bits represent a threshold number of entries (N) in the FIFO. When the FIFO fills to a point where it has more (N+1) than N words (data in FIFO exceeds the value set in the IDP_NSET bit field), a DAI interrupt is generated. This DAI interrupt corresponds to the IDP_FIFO_GTN_INT bit in DAI_IRPTL_x registers. Only the core can use this N threshold interrupt to detect when data needs to be read. The maximum IDP_NSET= 7, otherwise no interrupt is generated.
4	IDP_BHD	**IDP Buffer Hang Disable.** Reads of an empty FIFO or writes to a full FIFO to cause a core hang. This condition continues until the FIFO has valid data (in the case of reads) or the FIFO has at least one empty location (in the case of writes). Note this can be used in debug operations. 0 = Core hang is enabled 1 = Core hang is disabled
5	IDP_DMA_EN	**DMA Enable.** Enables DMA on all IDP channels. This bit is the global control for standard and ping-pong DMA. 0 = Channel disabled 1 = Channel enabled
6	IDP_CLROVR	**FIFO Overflow Clear Bit.** Writes of 1 to this bit clear the overflow condition in the DAI_STAT0 register. Because this is a WO-bit, it always returns 0 when read.
7	IDP_EN	**Enable IDP.** This bit enables the IDP. This is a global control bit. This bit needs to be set for all operations modes including DMA. When this bit is cleared (= 0), the IDP is disabled, and data cannot move to the IDP_FIFO. Note that when the IDP_EN bit transitions from 1 to 0, all data in the IDP FIFO are flushed. Writing a 1 to bit 31 of the IDP_CTL1 register also flushes the FIFO. This is a WO-bit and always returns a zero on reads.

Table A-19. IDP_CTL0 Register Bit Descriptions (Cont'd)

Bits	Name	Description
10–8	IDP_SMODE0	**Serial Input Data Format Mode Select.** These eight inputs (0-7), each of which contains 3 bits, indicate the mode of the serial input for each of the eight IDP channels. Input format: 000 = Left-justified mode 001 = I^2S mode 010 = Left-justified 32 bits 011 = I^2S 32 bits 100 = Right-justified 24 bits 101 = Right-justified 20 bits 110 = Right-justified 18 bits 111 = Right-justified 16 bits Note for I2S and left-justified single channel modes, it receives 32 bits of data from through SDATA pins. No L/R bit is added in these modes.
13–11	IDP_SMODE1	
16–14	IDP_SMODE2	
19–17	IDP_SMODE3	
22–20	IDP_SMODE4	
25–23	IDP_SMODE5	
28–26	IDP_SMODE6	
31–29	IDP_SMODE7	

Input Data Port Control Register 1 (IDP_CTL1)

Use the `IDP_CTL1` register to configure and enable individual IDP channels. The register is shown in Figure A-21 and described in Table A-20.

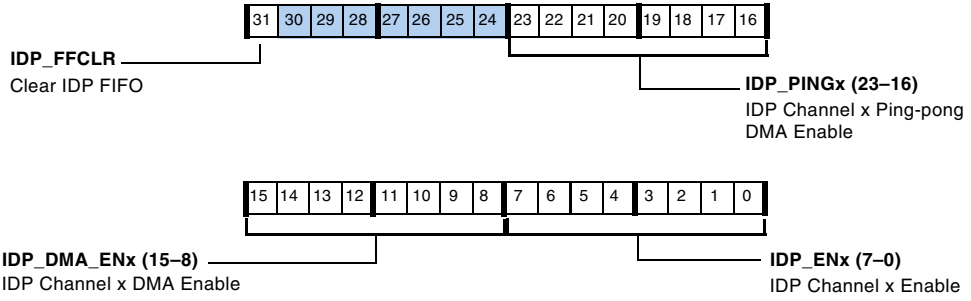

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
IDP_FFCLR
Clear IDP FIFO
IDP_PINGx (23–16)
IDP Channel x Ping-pong DMA Enable
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
IDP_DMA_ENx (15–8)
IDP Channel x DMA Enable
IDP_ENx (7–0)
IDP Channel x Enable

Figure A-21. IDP_CTL1 Register

Table A-20. IDP_CTL1 Register Bit Descriptions

Bit	Name	Description
7–0	IDP_ENx	**IDP Channel x Enable.** These are the enable bits for accepting data from individual channels. Corresponding IDP_ENx must be set with IDP_EN bit to get data from Channel x. If IDP_EN bit is not set then this bit has no effect. After RESET all these bits are disabled.
15–8	IDP_DMA_ENx	**IDP DMA Enable.** These are the DMA enable bits for individual channels. Corresponding IPD_DMA_ENx must be set with IDP_DMA_EN bit for DMA transfer of data from Channel x. If DMA_EN bit is not set then this bit has no effect. After RESET all these bits are enabled.
23–16	IDP_PINGx	**IDP ping-pong DMA Channel x Enable.** These are the Ping Pong DMA enable bits for individual channels. Corresponding IDP_PINGx must be set to start Ping Pong DMA from Channel x. This bit requires the IDP_DMA_ENx bit and IDP_DMA_EN bit are set. After RESET all these bits are enabled.
24	Reserved	
30–25	Reserved	
31	IDP_FFCLR	**Clear IDP FIFO.** Setting this bit to 1 clears the IDP FIFO. This is a write-only bit and always returns 0 on reads.

Parallel Data Acquisition Port Control Register (IDP_PP_CTL)

The IDP_PP_CTL register (shown in Figure A-22 and described in Table A-21) provides 20 mask bits that allow the input from any of the20 pins to be ignored.

For more information on the operation of the parallel data acquisition port, see “Parallel Data Acquisition Port (PDAP)” on page 8-9. For information on the pin muxing that is used in conjunction with this module, see “Pin Multiplexing” on page 14-17.

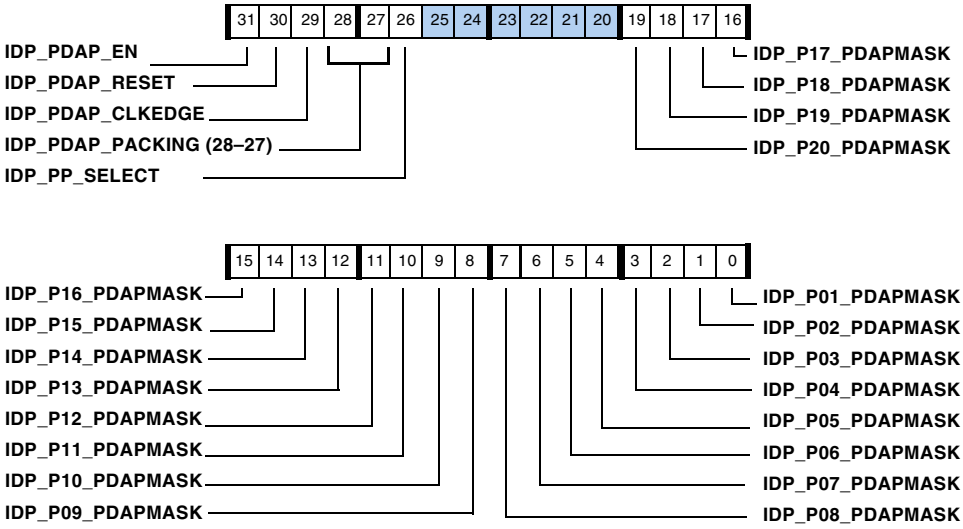


Figure A-22. IDP_PP_CTL Register

Table A-21. IDP_PP_CTL Register Bit Descriptions

Bit	Name	Description
19–0	IDP_P20–1_PDAPMASK	**Parallel Data Acquisition Port Mask.** For each of the parallel inputs, a bit is set (= 1) to indicate the bit is unmasked and therefore its data can be passed on to be read, or masked (= 0) so its data is not read. After this masking process, data gets passed along to the packing unit. 0 = Input data from PDAP_20-1 are masked 1 = Input data from PDAP_20-1 are unmasked
25–20	Reserved	
26	IDP_PP_SELECT	**PDAP Port Select.** This bit selects which pins are connected to the PDAP unit. 0 = Data/control bits are read from DAI pins 1 = Data/control bits are read from parallel port pins

Table A-21. IDP_PP_CTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
28–27	IDP_PDAP_PACKING	**Packing.** Selects PDAP packing mode. These bits mask parallel sub words from the 20 parallel input signals and packs them into a 32-bit word. The bit field indicates how data is packed. Selection of packing mode is made based on the application. 00 = 8- to 32-bit packing 01 = (11, 11, 10) to 32-bit packing 10 = 16- to 32-bit packing 11 = 20- to 32-bit packing. 12 LSBs are set to 0 Note for input data width less than 20-bits, inputs are aligned to MSB pins.
29	IDP_PDAP_CLKEDGE	**PDAP (Rising or Falling) Clock Edge.** Setting this bit (= 1) causes the data to latch on the falling edge (PDAP_CLK_I signal). Clearing this bit (= 0) causes data to latch on the rising edge (default). Notice that in all four packing modes described, data is read on a clock edge, but the specific edge used (rising or falling) is not indicated. 0 = Data is latched on the rising edge 1 = Data is latched on the falling edge
30	IDP_PDAP_RESET	**PDAP Reset.** A reset clears any data in the packing unit waiting to get latched into the IDP FIFO. This bit causes the reset circuit to strobe when asserted, and then automatically clears. Therefore, this bit always returns a value of zero when read.
31	IDP_PDAP_EN	**PDAP Enable.** 0 = Disconnects all PDAP inputs (data/control) from use as parallel input channel. 1 = Connects all PDAP inputs (data/control) from use as parallel input channel. IDP channel 0 cannot be used as a serial input port with this setting.

Input Data Port FIFO Register (IDP_FIFO)

The IDP_FIFO register (shown in Figure A-23 and described in Table A-22) provides information about the output of the 8-deep IDP FIFO. For more information, see "Data Buffer" on page 8-17.

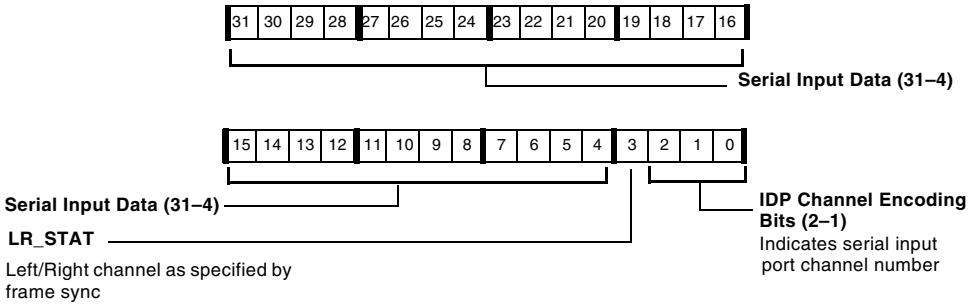

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
Serial Input Data (31–4)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Serial Input Data (31–4)
LR_STAT
Left/Right channel as specified by frame sync
IDP Channel Encoding Bits (2–1)
Indicates serial input port channel number

Figure A-23. IDP_FIFO Register

The information in Table A-22 is not valid when data comes from the PDAP channel.

Table A-22. IDP_FIFO Register Bit Descriptions

Bit	Name	Description
2–0	CHAN_ENC	**IDP Channel Encoding Bits.** Indicates serial input port channel number that provided this serial input data. Note: This information is not valid when data comes from the PDAP.
3	LR_STAT	**Left/Right Channel Status.** Indicates whether the data in bits 31-4 is the left or the right audio channel as dictated by the frame sync signal. The polarity of the encoding depends on the serial mode selected in IDP_SMODE for that channel. See Table A-19 on page A-48.
31–4	SDATA	**Input Data (Serial).** Some LSBs can be zero, depending on the mode.

IDP Status Register (DAI_STAT0)

The DAI_STAT0 register, shown Figure A-24 in and described in Table A-23 is a read-only register. The state of all eight DMA channels is reflected in the IDP_DMAx_STAT register (bits 24–17 of the DAI_STAT register). These bits are set once IDP_DMA_EN is set and remain set until the last data of that channel is transferred. Even if IDP_DMA_EN is set (=1) this bit goes low once the required number of data transfers occur. Furthermore, if DMA through some channel is not intended, itsIDP_DMAx_STAT bit goes high.

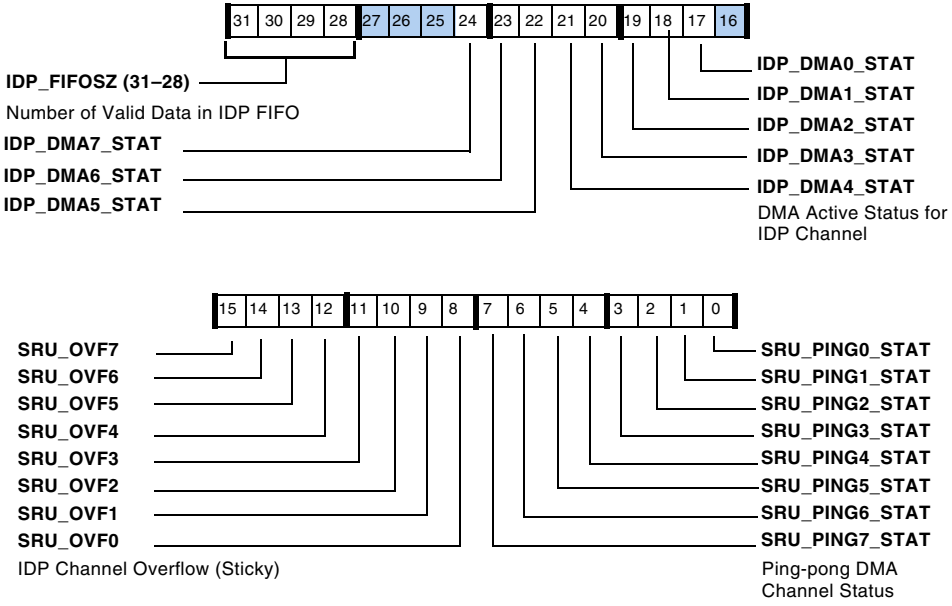


Figure A-24. DAI_STAT0 Register

Table A-23. DAI_STAT0 Register Bit Descriptions

Bit	Name	Description
7–0	SRU_PINGx_STAT	**Ping-pong DMA Channel Status.** Indicates the status of ping-pong DMA in each respective channel (channel 7–0). 0 = DMA is not active 1 = DMA is active
15–8	SRU_OVFx	**Sticky Overflow Channel Status.** Provides overflow status information for each channel (bit 8 for channel 0 through bit 15 for channel 7). 0 = SRU input no overflow 1 = SRU input overflow has occurred
16	Reserved	
24–17	IDP_DMAx_STAT	**Input Data Port DMA Channel Status.** 0 = DMA is not active 1 = DMA is active
27–25	Reserved	
31–28	IDP_FIFOSZ	**IDP FIFO Size.** Indicates Number of samples in the IDP FIFO. 0000 = IDP FIFO empty 1000 = IDP FIFO full

IDP Status Register 1 (DAI_STAT1)

Since the core does allow writes to the IDP_FIFO, the DAI_STAT1 register, described in Table A-24, stores the different read or writes indexes with a maximum of 8 entries each.

Table A-24. DAI_STAT1 Register Bit Descriptions

Bit	Name	Description
3–0	FIFO_WRI	Write Index Pointer. Reflects the write index status during core writes to the IDP_FIFO. 0000 = no write done 1000 = 8 writes done
7–4	FIFO_RDI	**Read Index Pointer.** Reflects the read index status during core reads from the IDP_FIFO. 0000 = no read done 1000 = 8 reads done
31–8	Reserved	

Peripheral Timer Registers

The timer peripheral module provides general-purpose timer functionality. It consists of three identical timer units. Each timer has memory-mapped registers described in the following sections. Each timer also has two 32-bit count registers decsribed in “Count Registers” on page 9-5.

Timer Configuration Registers (TMxCTL)

All timer clocks are gated off when the specific timer’s configuration register is set to zero at system reset or subsequently reset by user programs. These registers are shown in Figure A-25.

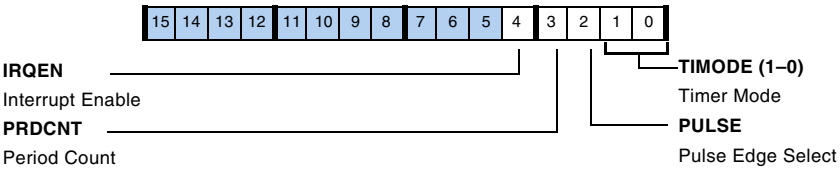


Figure A-25. TMxCTL Register

Table A-25. TMxCTL Register Bit Descriptions

Bit	Name	Definition
1–0	TIMODE	**Timer Mode.** 00 = Reset 01 = PWM_OUT mode (TIMODEPWM) 10 = WDTH_CAP mode (TIMODEW) 11 = EXT_CLK mode (TIMODEEXT)
2	PULSE	**Pulse Edge Select.** 1 = Positive active pulse 0 = Negative active pulse
3	PRDCNT	**Period Count.** 1 = Count to end of period 0 = Count to end of width
4	IRQEN	**Interrupt Enable.** 1 = Enable 0 = Disable

Timer Status Registers (TMxSTAT)

The global status registers `TMxSTAT` are shown in Figure A-26. Status bits are sticky and require a write-one to clear operation. During a status register read access, all reserved or unused bits return a zero. Each timer generates a unique processor interrupt request signal, `TIMxIRQ`.

A common status register latches these interrupts. Interrupt bits are sticky and must be cleared to assure that the interrupt is not reissued.

Each timer is provided with its own sticky status register TIMxEN bit. To enable or disable an individual timer, the TIMxEN bit is set or cleared. For example, writing a one to bit 8 sets the TIM0EN bit; writing a one to bit 9 clears it. Writing a one to both bit 8 and bit 9 clears TIM0EN. Reading the status register returns the TIM0EN state on both bit 8 and bit 9. The remaining TIMxEN bits operate similarly.

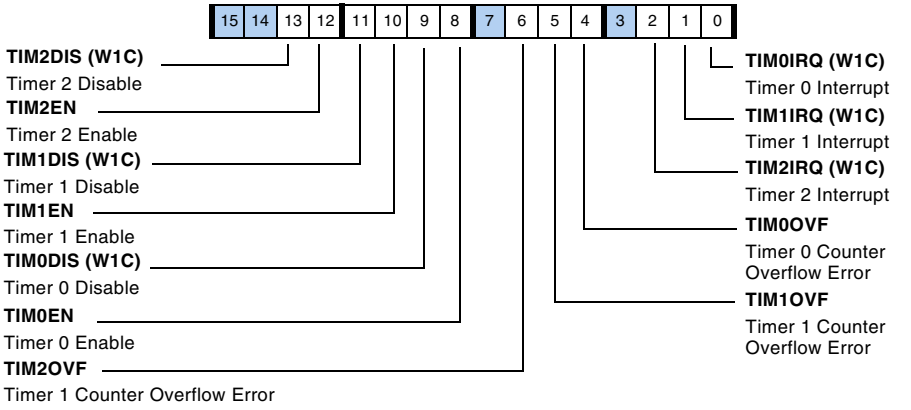


Figure A-26. TMxSTAT Register

Table A-26. TMxSTAT Register Bit Descriptions

Bit	Name	Description
0	TIM0IRQ Timer 0 Interrupt Latch	Write one-to-clear (also an output)
1	TIM1IRQ Timer 1 Interrupt Latch	Write one-to-clear (also an output)
2	TIM2IRQ Timer 2 Interrupt Latch	Write one-to-clear (also an output)
3	Reserved	
4	TIM0OVF Timer 0 Overflow/Error	Write one-to-clear (also an output)
5	TIM1OVF Timer 1 Overflow/Error	Write one-to-clear (also an output)
6	TIM2OVF Timer 2 Overflow/Error	Write one-to-clear (also an output)
7	Reserved	
8	TIM0EN Timer 0 Enable	Write one to enable timer 0

Table A-26. TMxSTAT Register Bit Descriptions (Cont'd)

Bit	Name	Description
9	TIM0EN Timer 0 Disable	Write one to disable timer 0
10	TIM1EN Timer 1 Enable	Write one to enable timer 1
11	TIM1EN Timer 1 Disable	Write one to disable timer 1
12	TIM2EN Timer 2 Enable	Write one to enable timer 2
13	TIM2EN Timer 2 Disable	Write one to disable timer 2
31–11	Reserved	

Sample Rate Converter Registers

The sample rate converter (SRC) is composed of five registers which are described in the following sections.

SRC Control Registers (SRCCTLx)

The `SRCCTLn` control registers (read/write) control the operating modes, filters, and data formats used in the sample rate converter. For n = 0, the register controls the SRC0 and SRC1 modules and for n = 1 it controls the SRC2 and SRC3 modules (x = 0, 2 andy = 1, 3). The bit settings for these registers are shown in Figure A-27 and described in Table A-27.

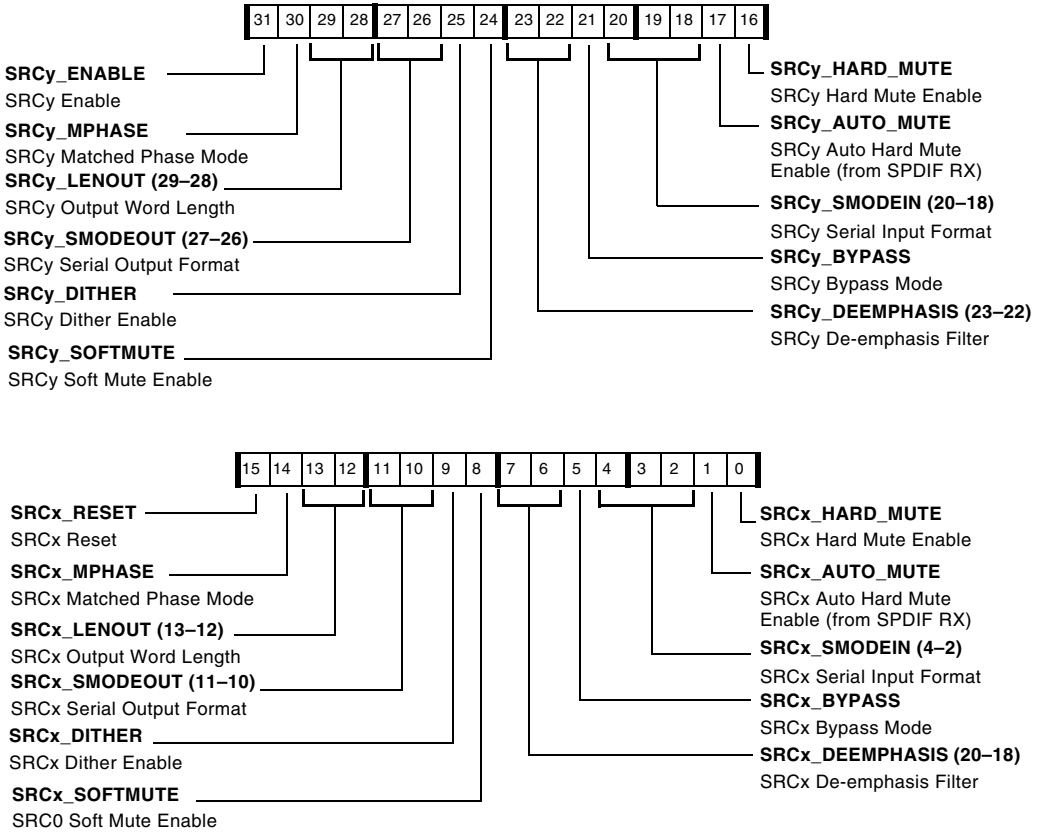


Figure A-27. SRCCTLn Register

Table A-27. SRCCTLn Register Bit Descriptions

Bit	Name	Description
0	SRCx_HARD_MUTE	**Hard Mute.** Hard mutes SRC 0, 2. 1 = Mute (default)
1	SRCx_AUTO_MUTE	**Auto Hard Mute.** Auto hard mutes SRC 0, 2 when non audio is asserted by the SPDIF receiver. 0 = No mute 1 = Mute (default)

Table A-27. SRCCTLn Register Bit Descriptions (Cont'd)

Bit	Name	Description
2–4	SRCx_SMODEIN	**Serial Input Format.** Selects the serial input format for SRC 0, 2 as follows: 000 = Default, format is left-justified 001 = I^2S 010 = TDM 100 = 24-bit right-justified 101 = 20-bit right-justified 110 = 18-bit right-justified 111 = 16-bit right-justified
5	SRCx_BYPASS	**Bypass SRCx.** Output of SRC 0, 2 is the same as the input.
6–7	SRCx_DEEMPHASIS	**De-emphasis Filter Select.** Enables de-emphasis on incoming audio data for SRC 0, 2. 00 = No de-emphasis 01 = 33 kHz 10 = 44.1 kHz 11 = 48 kHz
8	SRCx_SOFTMUTE	**Soft Mute.** Enables soft mute on SRC 0, 2. 0 = No mute 1 = Mute (default)
9	SRCx_DITHER	**Dither Select.** Enables dithering on SRC 0, 2 when a word length less than 24 bits is selected. 0 = Dithering is enabled (default) 1 = Dithering is disabled
10–11	SRCx_SMODEOUT	**Serial Output Format.** Selects the serial output format on SRC 0, 2 as follows: 00 = Left-justified (default) 01 = I^2S 10 = TDM mode 11 = Right-justified

Table A-27. SRCCTLn Register Bit Descriptions (Cont'd)

Bit	Name	Description
12–13	SRCx_LENOUT	**Output Word Length Select.** Selects the serial output word length on SRC 0, 2 as follows: 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits has dither added to the unused LSBs if SRCx_DITHER is enabled (= 1).
14	SRCx_MPHASE	**Match Phase Mode Select.** Configures the SRC 0, 2 modules to not use their own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data. 0 = Matched phase disabled (default) 1 = Matched phase enabled
15	SRCx_ENABLE	**SRC0 Enable.** Enables SRC 0, 2. 0 = Disabled 1 = Enabled
16	SRCy_HARD_MUTE	**Hard Mute.** Hard mutes SRC 1, 3. 1 = Mute (default)
17	SRCy_AUTO_MUTE	**Auto Hard Mute.** Auto hard mutes SRC 1, 3 when non audio is asserted by the SPDIF receiver. 0 = No mute 1 = Mute (default)
18–20	SRCy_SMODEIN	**Serial Input Format.** Selects the serial input format for SRC 1, 3 as follows: 000 = Default, format is left-justified 001 = I^2S 010 = TDM 100 = 24-bit right-justified 101 = 20-bit right-justified 110 = 18-bit right-justified 111 = 16-bit right-justified
21	SRCy_BYPASS	**Bypass Mode Enable.** Output of SRC 1, 3 is the same as input.

Table A-27. SRCCTLn Register Bit Descriptions (Cont'd)

Bit	Name	Description
22–23	SRCy_DEEMPHASIS	**De-emphasis Filter Select.** Enables de-emphasis on incoming audio data for SRC 1, 3. 00 = No de-emphasis 01 = 33 kHz 10 = 44.1 kHz 11 = 48 kHz
24	SRCy_SOFTMUTE	**Soft Mute.** Enables soft mute on SRC 1, 3. 0 = No mute 1 = Mute (default)
25	SRCy_DITHER	**Dither Select.** Disables dithering on SRC 1, 3 when a word length less than 24 bits is selected. 0 = Dithering is disabled (default) 1 = Dithering is enabled
26–27	SRCy_SMODEOUT	**Serial Output Format.** Selects the serial output format for SRC 1, 3 as follows. 00 = Left-justified (default) 01 = I^2S 10 = TDM mode 11 = Right-justified
28–29	SRCy_LENOUT	**Output Word Length Select.** Selects the serial output word length for SRC 1, 3 as follows: 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits has dither added to the unused LSBs if SRCx_DITHER is enabled (= 0).
30	SRCy_MPHASE	**Match Phase Mode Select.** Configures the SRC 1, 3 modules to not use their own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data. 0 = Matched phase disabled (default) 1 = Matched phase enabled
31	SRCy_ENABLE	**SRC Enable.** Enables SRC 1, 3. 0 = Disabled 1 = Enabled

SRC Mute Register (SRCMUTE)

This read/write register connects an SRCx mute input and output when the SRCO_MUTE_ENx bit is cleared (=0). This allows SRCx to automatically mute input while the SRC is initializing (0 = automatic muting and 1 = manual muting). Bit 0 controls SRC0, bit 1 controls SRC1, bit 2 controls SRC2, and bit 3 controls SRC3.

SRC Ratio Registers (SRCRATx)

These read-only status registers report the mute and I/O sample ratio as follows: the SRCRAT0 register reports for SRC0 and SRC1 and the SRCRAT1 register reports the mute and I/O sample ratio for SRC2 and SRC3. The registers are shown in Figure A-28 and Figure A-29.

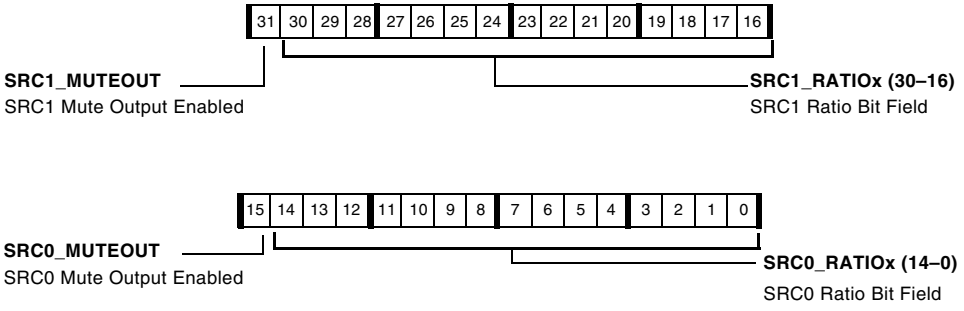

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SRC1_MUTEOUT
SRC1 Mute Output Enabled
SRC1_RATIOx (30–16)
SRC1 Ratio Bit Field
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SRC0_MUTEOUT
SRC0 Mute Output Enabled
SRC0_RATIOx (14–0)
SRC0 Ratio Bit Field

Figure A-28. SRC Ratio Register (SRCRAT0)

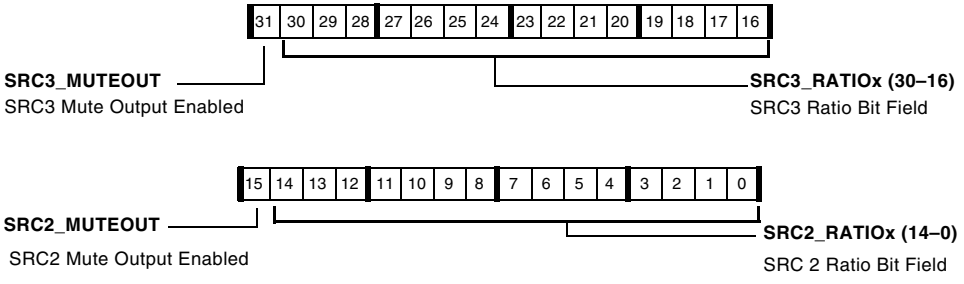

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SRC3_MUTEOUT
SRC3 Mute Output Enabled
SRC3_RATIOx (30–16)
SRC3 Ratio Bit Field
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SRC2_MUTEOUT
SRC2 Mute Output Enabled
SRC2_RATIOx (14–0)
SRC 2 Ratio Bit Field

Figure A-29. SRC Ratio Register (SRCRAT1)

Precision Clock Generator Registers

The precision clock generator (PCG) consists of two identical units. Each of these two units (A and B) generates one clock (CLKA_O or CLKB_O) and one frame sync (FSA_O or FSB_O) output. These units can take an input clock signal from the CLKIN source (crystal/oscillator) or any of the sources in group A of the signal routing unit. These signals are controlled by seven memory-mapped registers described in the following sections.

Control Registers (PCG_CTLxy)

These registers, shown in Figure A-30 and Figure A-31 and described in Table A-28 and Table A-29, configure and enable the PCGs. Note the frame sync phase counter setting are split between control registers (PHASE_LO or PHASE_HI). In the tables and figures, x = PCG A and B.

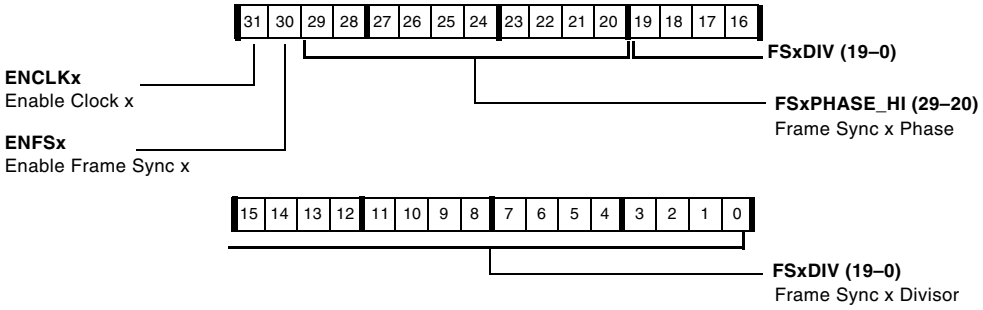

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
FSxDIV (19–0)
ENCLKx
Enable Clock x
FSxPHASE_HI (29–20)
Frame Sync x Phase
ENFSx
Enable Frame Sync x
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
FSxDIV (19–0)
Frame Sync x Divisor

Figure A-30. PCG_CTLx0 Register

Table A-28. PCG_CTLx0 Register Bit Descriptions

Bit	Name	Description
19–0	FSxDIV	**Divisor for Frame Sync x.**
29–20	FSxPHASE_HI	**Phase for Frame Sync x.** This field represents the upper half of the 20-bit value for the channel x frame sync phase. See also FSxPHASE_LO (bits 29–20) in PCG_CTLx1 described on on page A-66.
30	ENFSx	**Enable Frame Sync A.** 0 = Frame sync x generation disabled 1 = Frame sync x generation enabled
31	ENCLKx	**Enable Clock A.** 0 = Clock x generation disabled 1 = Clock x generation enabled

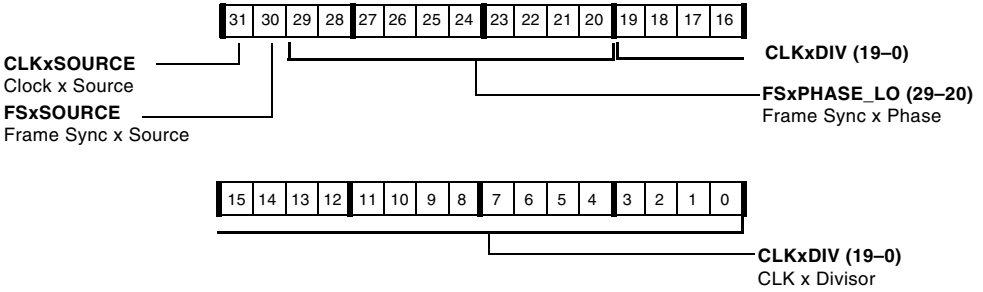

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
CLKxSOURCE
Clock x Source
FSxSOURCE
Frame Sync x Source
CLKxDIV (19–0)
FSxPHASE_LO (29–20)
Frame Sync x Phase
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
CLKxDIV (19–0)
CLK x Divisor

Figure A-31. PCG_CTLx1 Register

Table A-29. PCG_CTLx1 Register Bit Descriptions

Bit	Name	Description
19–0	CLKxDIV	**Divisor for Clock x.**
29–20	FSxPHASE_LO	**Phase for Frame Sync x.** This field represents the lower half of the 20-bit value for the channel x frame sync phase. See also FSxPHASE_HI (bits 29-20) in PCG_CTLx0 shown in on page A-66.

Table A-29. PCG_CTLx1 Register Bit Descriptions (Cont'd)

Bit	Name	Description
30	FSxSOURCE	**Frame Sync x Source.** Master clock source for frame sync x. 0 = CLKIN source selected for clock x 1 = PCG_EXTA_I (SRU) selected for frame sync x
31	CLKxSOURCE	**Clock x Source.** Master clock source for clock x. 0 = CLKIN source selected for clock x 1 = PCG_EXTA_I (SRU) selected for clock x

Pulse Width Register (PCG_PW)

These registers, shown in Figure A-32 and Figure A-33 and described in Table A-30 and Table A-31, are used to set the pulse width which is the number of input clock periods for which the frame sync output is HIGH.

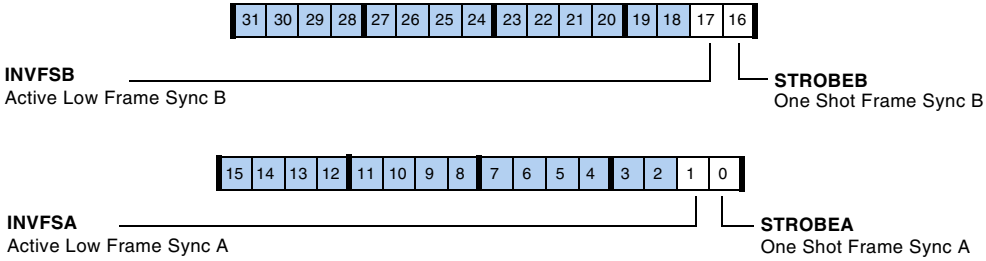

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
INVFSB
Active Low Frame Sync B
STROBEB
One Shot Frame Sync B
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
INVFSA
Active Low Frame Sync A
STROBEA
One Shot Frame Sync A

Figure A-32. PCG_PW Register (Bypass Mode)

Table A-30. PCG_PW Register Bit Descriptions (Bypass Mode)

Bit	Name	Description
0	STROBEA	**One Shot Frame Sync A.** Frame sync is a pulse with a duration equal to one period of the MISCA2_I signal repeating at the beginning of every frame. Note: This is valid in bypass mode only.
1	INVFSA	**Active Low Frame Sync Select for Frame Sync A.** 0 = Active high frame sync 1 = Active low frame sync

Table A-30. PCG_PW Register Bit Descriptions (Bypass Mode) (Cont'd)

Bit	Name	Description
15–2	Reserved (in bypass mode, bits 15–2 are ignored).	
16	STROBEB	**One Shot Frame Sync B.** Frame sync is a pulse with a duration equal to one period of the MISCA3_I signal repeating at the beginning of every frame. Note: This is valid in bypass mode only.
17	INVFSB	**Active Low Frame Sync Select.** 0 = Active high frame sync 1 = Active low frame sync
31–18	Reserved (in bypass mode, bits 31–18 are ignored).	

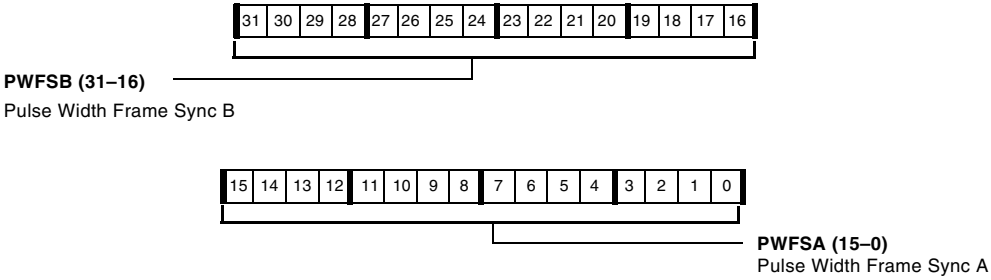


Figure A-33. PCG_PW Register (Normal Mode)

Table A-31. PCG_PW Register Bit Descriptions (Normal Mode)

Bit	Name	Description
15–0	PWFSA	**Pulse Width for Frame Sync A.** Note: This is valid when not in bypass mode.
31–16	PWFSB	**Pulse Width for Frame Sync B.** Note: This is valid when not in bypass mode.

Synchronization Register (PCG_SYNC)

This register, in conjunction with the control registers, allows the frame sync output to be synchronized with an external clock. This register is shown in Figure A-34 and described in Table A-32.

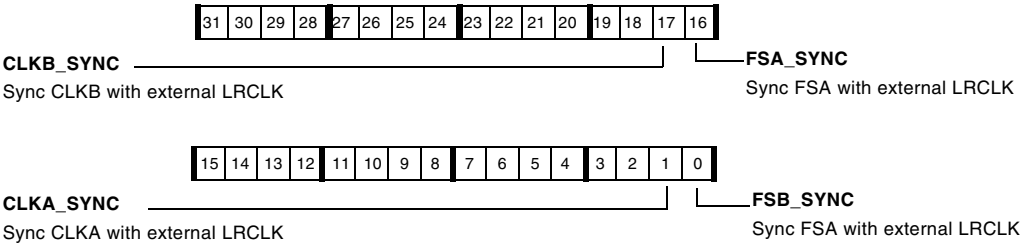

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
CLKB_SYNC
Sync CLKB with external LRCLK
FSA_SYNC
Sync FSA with external LRCLK
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
CLKA_SYNC
Sync CLKA with external LRCLK
FSB_SYNC
Sync FSA with external LRCLK

Figure A-34. PCG_SYNC Register

Table A-32. PCG_SYNC Register Bit Descriptions (in Normal Mode)

Bit	Name	Description
0	FSA_SYNC	Enable trigger of frame sync A with external LRCLK
1	CLKA_SYNC	Enable trigger of CLKA with external LRCLK
16	FSB_SYNC	Enable trigger of frame sync B with external LRCLK
17	CLKB_SYNC	Enable trigger of CLKB with external LRCLK

Sony/Philips Digital Interface Registers

The following sections describe the registers that are used to configure, enable, and report status information for the S/PDIF transceiver.

Transmitter Registers

Transmit Control Register (DITCTL)

This 32-bit read/write register's bits are shown in Figure A-35 and described in Table A-33.

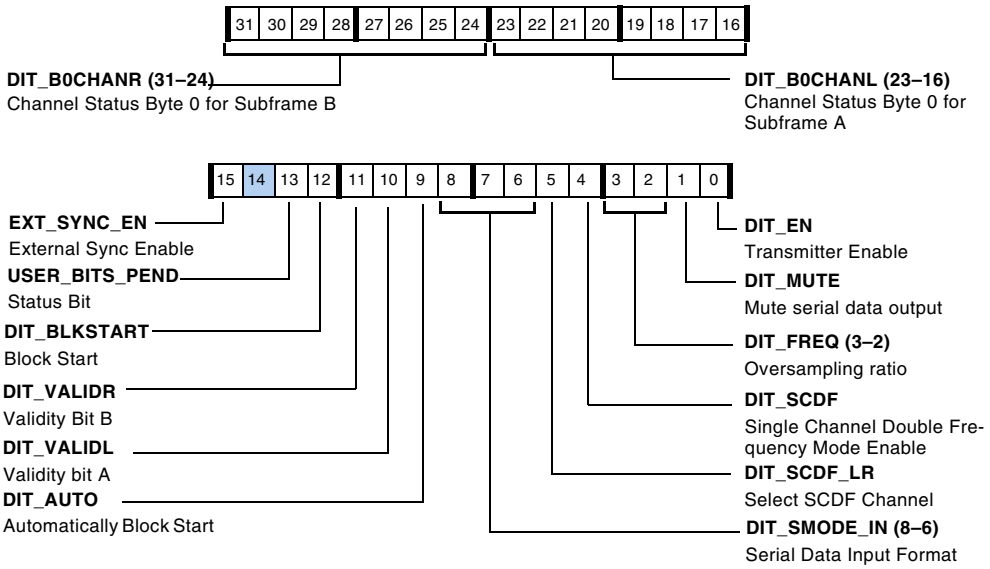


Figure A-35. DITCTL Register

Table A-33. DITCTL Register Bit Descriptions

Bit	Name	Description
0	DIT_EN	**Transmitter Enable.** Enables the transmitter and resets the control registers to their defaults. 0 = Transmitter disabled 1 = Transmitter enabled
1	DIT_MUTE	**Mute.** Mutes the serial data output.
3–2	DIT_FREQ	**Frequency Multiplier.** Sets the over sampling ratio to the following: 00 = 256 x frame sync 01 = 384 x frame sync 10, 11 = Reserved
4	DIT_SCDF	**Single-channel, Double-frequency Mode Enable.** 0 = 2 channel mode 1 = SCDF mode
5	DIT_SCDF_LR	**Select Single-channel, Double-frequency Mode.** 0 = Left channel 1 = Right channel
8–6	DIT_SMODEIN	**Serial Data Input Format.** Selects the input format as follows: 000 = Left-justified 001 = I^2S 010 = reserved 011 = reserved 100 = Right-justified, 24-bits 101 = Right-justified, 20-bits 110 = Right-justified, 18-bits 111 = Right-justified, 16-bits
9	DIT_AUTO	**Automatically Generate Block Start.** Automatically generate block start. When enabled, the transmitter is in standalone mode where it inserts block start, channel status, and validity bits on its own. 0 = Manually start block transfer according to input stream status bits 1 = Automatically start block transfer.
10	DIT_VALIDL	**Validity Bit A.** Use with channel status buffer.
11	DIT_VALIDR	**Validity Bit B.** Use with channel status buffer.

Table A-33. DITCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
12	DIT_BLKSTART	**Block Start (read-only).** Status bit that indicates block start (when bit 9, DIT_AUTO = 1). 0 = Current word is not block start 1 = Current word is block start
13	USER_BITS_PEND	This is a status bit and is high if user bit buffer has been written but not completely transmitted.
14	Reserved	
15	EXT_SYNC_EN	**External Sync Enable.** When set, frame counter is set to zero at an LRCLK rising edge following EXT_SYNC_I rising edge.
23–16	DIT_B0CHANL	Channel status byte 0 for subframe A
31–24	DIT_B0CHANR	Channel status byte 0 for subframe B

Transmit Status Registers for Subframe A (DITCHANL)

The S/PDIF transmitter stores a maximum of 5 bytes (40-bit) status information. Note that status byte 0 is available in the `DITCTL` register. This 32-bit read/write register is described in Table A-34.

Table A-34. Status Registers

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCTL			BYTE0	
DITCHANL0 BY	TE1	BYTE2	BYTE3	BYTE4

Transmit Status Registers for Subframe B (DITCHANR)

The S/PDIF transmitter stores a maximum of 5 bytes (40-bit) status information. Note that status byte 0 is available in the DITCTL register. This 32-bit read/write register is described in Table A-35.

Table A-35. Status Registers

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCTL				BYTE0
DITCHANR	BYTE1	BYTE2	BYTE3	BYTE4

Receiver Registers

Receive Control Register (DIRCTL)

This 32-bit, read/write register, described in Table A-36 is used to set up error control and single-channel double-frequency mode.

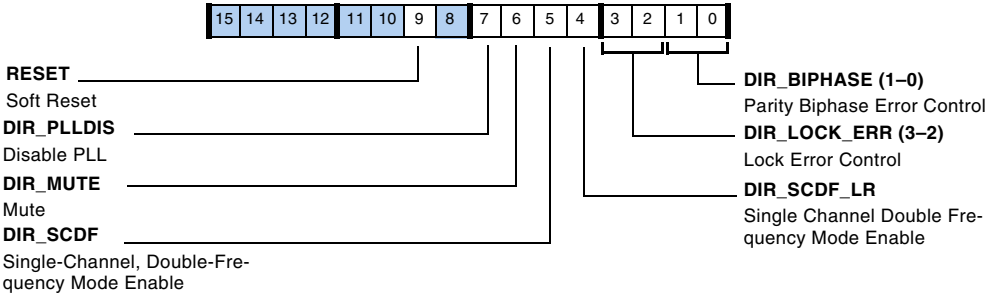


Figure A-36. DIRCTL Register

Table A-36. DIRCTL Register Bit Descriptions

Bit	Name	Description
1–0	DIR_BIPHASE	**Parity Biphase Error Control.** 00 = No action taken 01 = Hold last valid sample 10 = Replace invalid sample with zeros 11 = Reserved
3–2	DIR_LOCK_ERR	**Lock Error Control.** 00 = No action taken 01 = Hold last valid sample 10 = Send zeros after the last valid sample 11 = Soft mute of the last valid audio sample performs as if NOSTREAM is asserted. This is valid only when linear PCM audio data is in the stream. When non-linear audio data is in the stream, this mode defaults to the case of DIR_ERR_CTL = 10.
4	DIR_SCDF_LR	**Single-channel, Double-frequency Channel Select.** 0 = Left channel 1 = Right channel
5	DIR_SCDF	**Single-channel, Double-frequency Mode Enable.** 0 = 2 channel mode enabled 1 = SCDF mode
6	DIR_MUTE	**Mute.** 0 = Mute disabled 1 = Mute serial data outputs, maintaining clocks (digital black)
7	DIR_PLLDIS	**Disable PLL.** Determines clock input. 0 = Use derived clock from the digital PLL 1 = Use clock input from external PLL
8	Reserved	
9	RESET	**Reset S/PDIF** For soft resetting the SPDIF receiver. PLL does not reset with this bit.
31–10	Reserved	

Receive Status Register (DIRSTAT)

This 32-bit, read-only register is used to store the error bits. The error bits are sticky on read. Once they are set, they remain set until the register is read. This register also contains the lower byte of the 40-bit channel status information. The bit settings for these registers are shown in Figure A-37 and described in Table A-37.

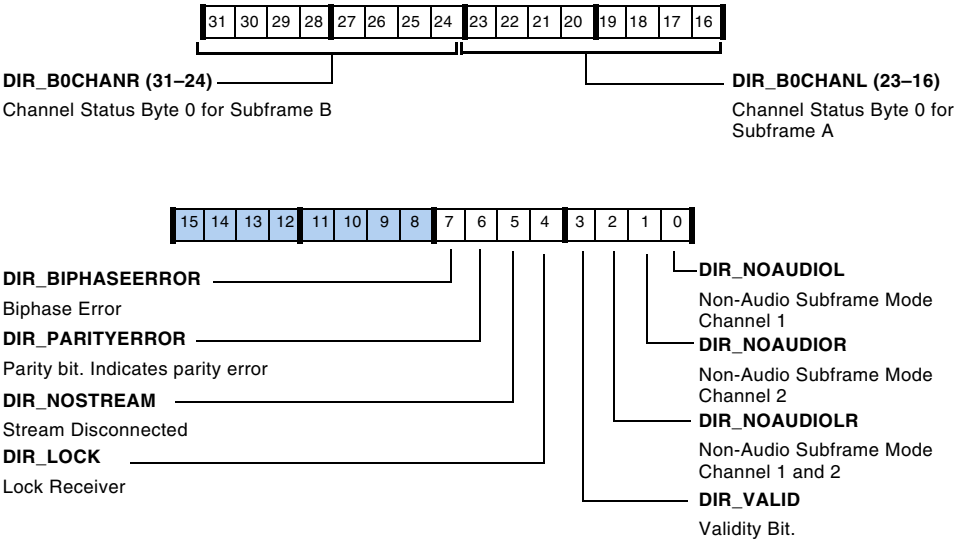

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
DIR_B0CHANR (31–24)
Channel Status Byte 0 for Subframe B
DIR_B0CHANL (23–16)
Channel Status Byte 0 for Subframe A
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DIR_BIPHASEERROR
Biphase Error
DIR_PARITYERROR
Parity bit. Indicates parity error
DIR_NOSTREAM
Stream Disconnected
DIR_LOCK
Lock Receiver
DIR_NOAUDIOL
Non-Audio Subframe Mode Channel 1
DIR_NOAUDIOR
Non-Audio Subframe Mode Channel 2
DIR_NOAUDIOLR
Non-Audio Subframe Mode Channel 1 and 2
DIR_VALID
Validity Bit.

Figure A-37. DIRSTAT Register

Table A-37. DIRSTAT Register Bit Descriptions

Bit	Name	Description
0	DIR_NOAUDIOL	**Non-Audio Subframe Mode Channel 1.** Based on SMPTE 337M. 0 = Not non-audio subframe mode 1 = Non-audio subframe mode, channel 1
1	DIR_NOAUDIOR	**Non-Audio Subframe Mode Channel 2.** Based on SMPTE 337M. 0 = Not non-audio subframe mode 1 = Non-audio subframe mode, channel 2
2	DIR_NOAUDIOLR	**Non-Audio Frame Mode Channel 1 and 2.** Based on SMPTE 337M. 0 = Not non-audio frame mode 1 = Non-audio frame mode
3	DIR_VALID	**Validity Bit.** ORed value of channel 1 and 2. 0 = Linear PCM data 1 = Non-linear audio data
4	DIR_LOCK	**Lock Receiver.** 0 = Receiver not locked 1 = Receiver locked
5	DIR_NOSTREAM	**Stream Disconnected.** Indicates that the data stream is disconnected. 0 = Stream not disconnected 1 = Stream disconnected
6	DIR_PARITYERROR	**Parity Bit.** Indicates parity error. 0 = No parity error 1 = Parity error
7	DIR_BIPHASEERROR	**Biphase Error.** Indicates biphase error. 0 = No biphase error 1 = Biphase error
15–8	Reserved	
23–16	DIR_B0CHANL	**Channel Status Byte 0 for Subframe A.**
31–24	DIR_B0CHANR	**Channel Status Byte 0 for Subframe B.**

Receive Status Registers for Subframe A (DIRCHANL)

The S/PDIF receiver stores a maximum of 5 bytes (40-bit) status information. Note that status byte 0 is available in the DIRCTL register. This 32-bit read/write register is described in Table A-38.

Table A-38. Status Registers

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DIRSTAT			BYTE0	
DIRCHANL	BYTE1	BYTE2	BYTE3	BYTE4

Receive Status Registers for Subframe B (DIRCHANR)

The S/PDIF receiver stores a maximum of 5 bytes (40-bit) status information. Note that status byte 0 is available in the DIRCTL register. This 32-bit read/write register is described in Table A-39.

Table A-39. Status Registers

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DIRSTAT				BYTE0
DIRCHANR	BYTE1	BYTE2	BYTE3	BYTE4

DAI Interrupt Controller Registers

All of the DAI interrupt registers are used primarily to provide the status of the resident interrupt controller. These registers are listed in Table A-40 and shown in Figure A-38. Note that for each of these registers the bit names and numbers are the same.

Table A-40. DAI Interrupt Registers

Register	Description
DAI_IRPTL_H DAI_IRPTL_HS	High priority interrupt latch register Shadow high priority interrupt latch register
DAI_IRPTL_L DAI_IRPTL_LS	Low priority interrupt latch register Shadow low priority interrupt latch register
DAI_IRPTL_PRI	Core interrupt priority assignment register
DAI_IRPTL_RE	Rising edge interrupt mask register
DAI_IRPTL_FE	Falling edge interrupt mask register

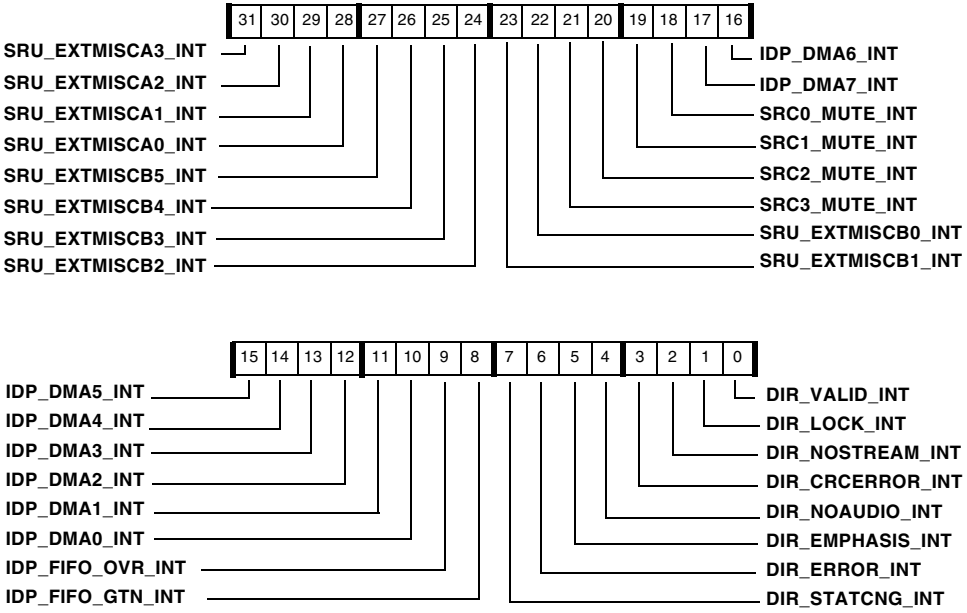


Figure A-38. DAI_IRPTL_x Register

DAI Status Register

The DAI_STAT register, shown in Figure A-39 and described in Table A-41, and the DAI_PIN_STAT register, shown in Figure A-67 on

page A-107, provide status information for the IDP/PDAP DMA channels. The DAI_PIN_PULLUP register, shown in Figure A-66, allows programs to enable/disable pull-up resistors.

Digital Applications Interface Status Register (DAI_STAT)

The DAI_STAT register is a read-only register. The state of all eight DMA channels is reflected in the IDP_DMAx_STAT register (bits 24–17 of the DAI_STAT register). These bits are set onceIDP_DMA_EN is set and remain set until the last data of that channel is transferred. Even if IDP_DMA_EN is set (=1) this bit goes low once the required number of data transfers occur. Furthermore, if DMA through some channel is not intended, its IDP_DMAx_STAT bit goes high.

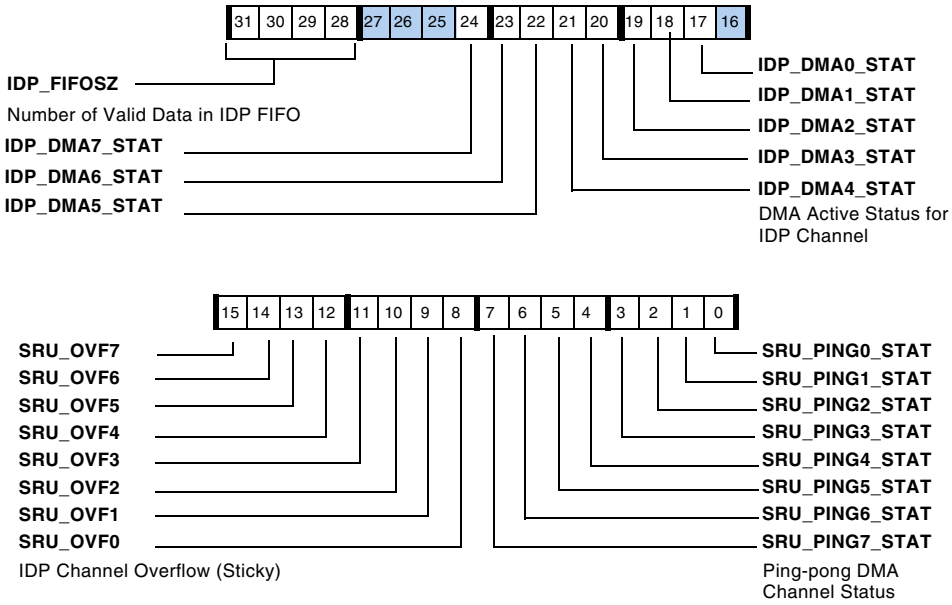


Figure A-39. DAI_STAT Register

Table A-41. DAI_STAT Register Bit Descriptions

Bit	Name	Description
7–0	SRU_PINGx_STAT	**Ping-pong DMA Status (Channel).** Indicates the status of ping-pong DMA in each respective channel (channel 7–0). 0 = DMA is not active 1 = DMA is active
15–8	SRU_OVFx	**Sticky Overflow Status (Channel).** Provides overflow status information for each channel (bit 8 for channel 0 through bit 15 for channel 7). 0 = No overflow 1 = Overflow has occurred
16	Reserved	
24–17	IDP_DMAx_STAT	**Input Data Port DMA Status.** 0 = DMA is not active 1 = DMA is active
27–25	Reserved	
31–28	IDP_FIFOSZ	**IDP FIFO Size.** 0000 = IDP FIFO empty 1000 = IDP FIFO full

DAI Signal Routing Unit Registers

The digital applications interface (DAI) is comprised of a group of peripherals and the signal routing unit (SRU).

Clock Routing Control Registers (SRU_CLKx, Group A)

These registers (see Figure A-40 through Figure A-44) correspond to the group A clock sources listed in Table A-42. Each of the clock inputs are connected to a clock source, based onthe 5-bit values in the figures. When either of the precision clock generators is used in external source mode, the `SRU_CLK3` register, pins 0–4 and/or pins 5–9, specify the source.

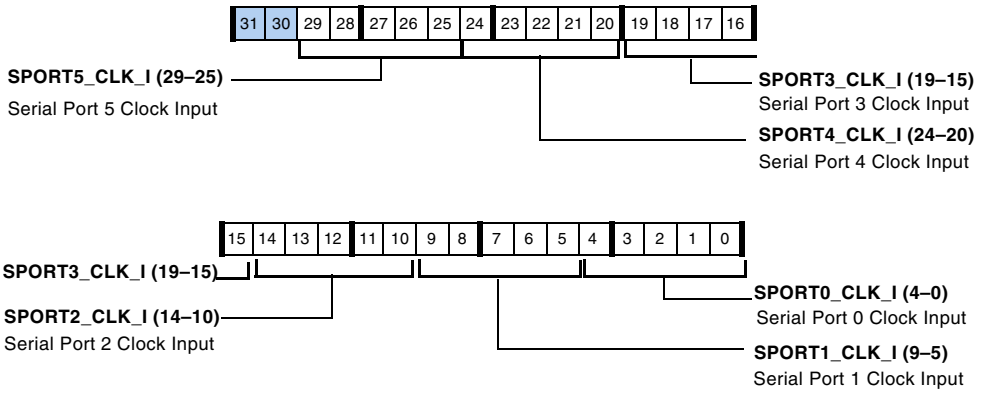

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SPORT5_CLK_I (29–25)
Serial Port 5 Clock Input
SPORT3_CLK_I (19–15)
Serial Port 3 Clock Input
SPORT4_CLK_I (24–20)
Serial Port 4 Clock Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SPORT3_CLK_I (19–15)
SPORT2_CLK_I (14–10)
Serial Port 2 Clock Input
SPORT0_CLK_I (4–0)
Serial Port 0 Clock Input
SPORT1_CLK_I (9–5)
Serial Port 1 Clock Input

Figure A-40. SRU_CLK0 Register

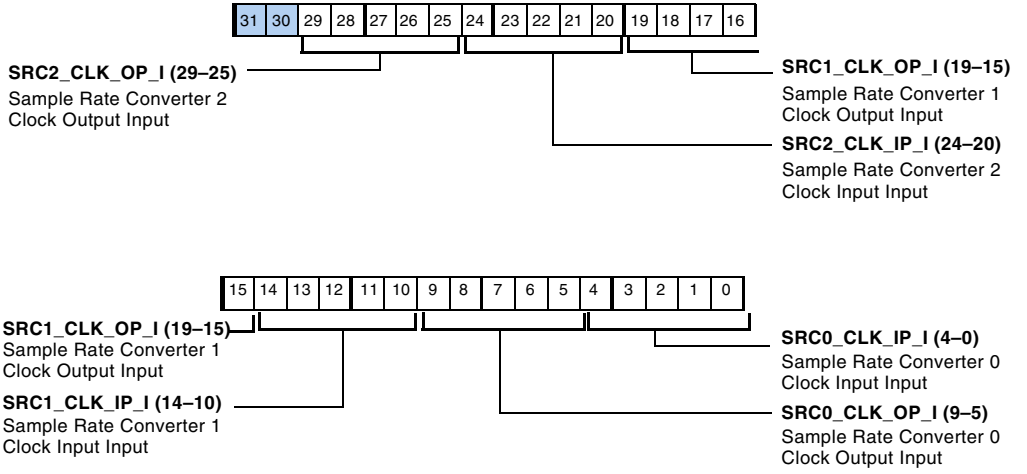

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SRC2_CLK_OP_I (29–25)
Sample Rate Converter 2
Clock Output Input
SRC1_CLK_OP_I (19–15)
Sample Rate Converter 1
Clock Output Input
SRC2_CLK_IP_I (24–20)
Sample Rate Converter 2
Clock Input Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SRC1_CLK_OP_I (19–15)
Sample Rate Converter 1
Clock Output Input
SRC1_CLK_IP_I (14–10)
Sample Rate Converter 1
Clock Input Input
SRC0_CLK_IP_I (4–0)
Sample Rate Converter 0
Clock Input Input
SRC0_CLK_OP_I (9–5)
Sample Rate Converter 0
Clock Output Input

Figure A-41. SRU_CLK1 Register

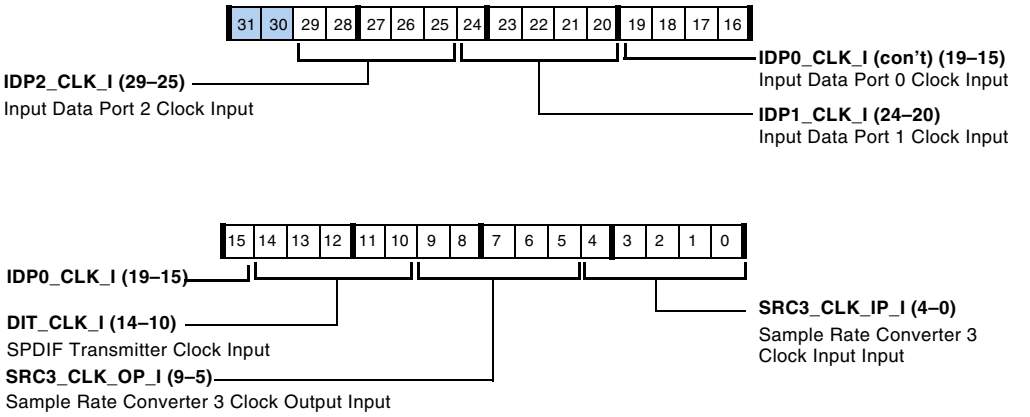

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
IDP2_CLK_I (29–25)
Input Data Port 2 Clock Input
IDP0_CLK_I (con't) (19–15)
Input Data Port 0 Clock Input
IDP1_CLK_I (24–20)
Input Data Port 1 Clock Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
IDP0_CLK_I (19–15)
DIT_CLK_I (14–10)
SPDIF Transmitter Clock Input
SRC3_CLK_OP_I (9–5)
Sample Rate Converter 3 Clock Output Input
SRC3_CLK_IP_I (4–0)
Sample Rate Converter 3
Clock Input Input

Figure A-42. SRU_CLK2 Register

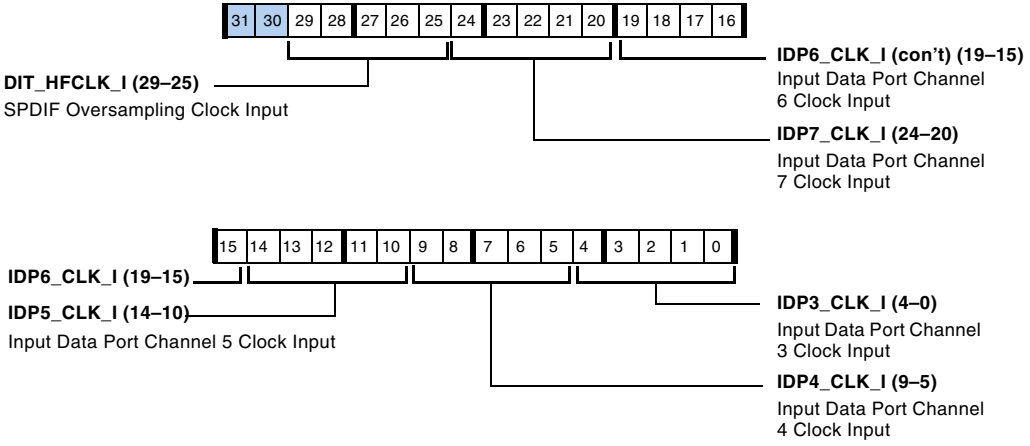
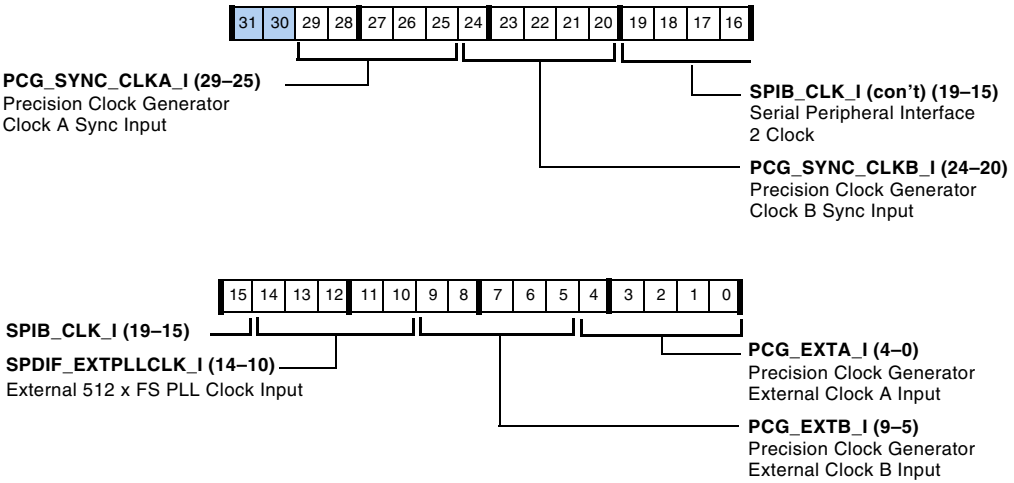

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
DIT_HFCLK_I (29–25)
SPDIF Oversampling Clock Input
IDP6_CLK_I (con't) (19–15)
Input Data Port Channel
6 Clock Input
IDP7_CLK_I (24–20)
Input Data Port Channel
7 Clock Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
IDP6_CLK_I (19–15)
IDP5_CLK_I (14–10)
Input Data Port Channel 5 Clock Input
IDP3_CLK_I (4–0)
Input Data Port Channel
3 Clock Input
IDP4_CLK_I (9–5)
Input Data Port Channel
4 Clock Input

Figure A-43. SRU_CLK3 Register


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
PCG_SYNC_CLKA_I (29–25)
Precision Clock Generator
Clock A Sync Input
SPIB_CLK_I (con't) (19–15)
Serial Peripheral Interface
2 Clock
PCG_SYNC_CLKB_I (24–20)
Precision Clock Generator
Clock B Sync Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SPIB_CLK_I (19–15)
SPDIF_EXTPLLCLK_I (14–10)
External 512 x FS PLL Clock Input
PCG_EXTA_I (4–0)
Precision Clock Generator
External Clock A Input
PCG_EXTB_I (9–5)
Precision Clock Generator
External Clock B Input

Setting SRU_CLK4 4–0 = 28 connects PCG_EXTA_I to logic low, not to PCG_CLKA_O.
Setting SRU_CLK4 9–5 = 29 connects PCG_EXTB_I to logic low, not to PCG_CLKB_O.

Figure A-44. SRU_CLK4 Register

Table A-42. Group A Sources – Serial Clock

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI_PB01_O	Pin buffer 1
00001 (0x1)	DAI_PB02_O	Pin buffer 2
00010 (0x2)	DAI_PB03_O	Pin buffer 3
00011 (0x3)	DAI_PB04_O	Pin buffer 4
00100 (0x4)	DAI_PB05_O	Pin buffer 5
00101 (0x5)	DAI_PB06_O	Pin buffer 6
00110 (0x6)	DAI_PB07_O	Pin buffer 7
00111 (0x7)	DAI_PB08_O	Pin buffer 8
01000 (0x8)	DAI_PB09_O	Pin buffer 9
01001 (0x9)	DAI_PB10_O	Pin buffer 10
01010 (0xA)	DAI_PB11_O	Pin buffer 11

Table A-42. Group A Sources – Serial Clock (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
01011 (0xB)	DAI_PB12_O	Pin buffer 12
01100 (0xC)	DAI_PB13_O	Pin buffer 13
01101 (0xD)	DAI_PB14_O	Pin buffer 14
01110 (0xE)	DAI_PB15_O	Pin buffer 15
01111 (0xF)	DAI_PB16_O	Pin buffer 16
10000 (0x10)	DAI_PB17_O	Pin buffer 17
10001 (0x11)	DAI_PB18_O	Pin buffer 18
10010 (0x12)	DAI_PB19_O	Pin buffer 19
10011 (0x13)	DAI_PB20_O	Pin buffer 20
10100 (0x14)	SPORT0_CLK_O	SPORT 0 clock
10101 (0x15)	SPORT1_CLK_O	SPORT 1 clock
10110 (0x16)	SPORT2_CLK_O	SPORT 2 clock
10111 (0x17)	SPORT3_CLK_O	SPORT 3 clock
11000 (0x18)	SPORT4_CLK_O	SPORT 4 clock
11001 (0x19)	SPORT5_CLK_O	SPORT 5 clock
11010 (0x1A)	DIR_CLK_O	SPDIF receive clock output
11011 (0x1B)	DIR_TDMCLK_O	SPDIF receive TDM clock output
11100 (0x1C)	PCG_CLKA_O	Precision clock A output
11101 (0x1D)	PCG_CLKB_O	Precision clock B output
11110 (0x1E)	LOW	Logic level low (0)
11111 (0x1F)	HIGH	Logic level high (1)

Serial Data Routing Registers (SRU_DATx, Group B)

The serial data routing control registers (see Figure A-46 through Figure A-50) route serial data to the serial ports (A and B data channels) and the input data port. Each of the data inputs specified are connected to a data source based on the 6-bit values shown in Table A-43.

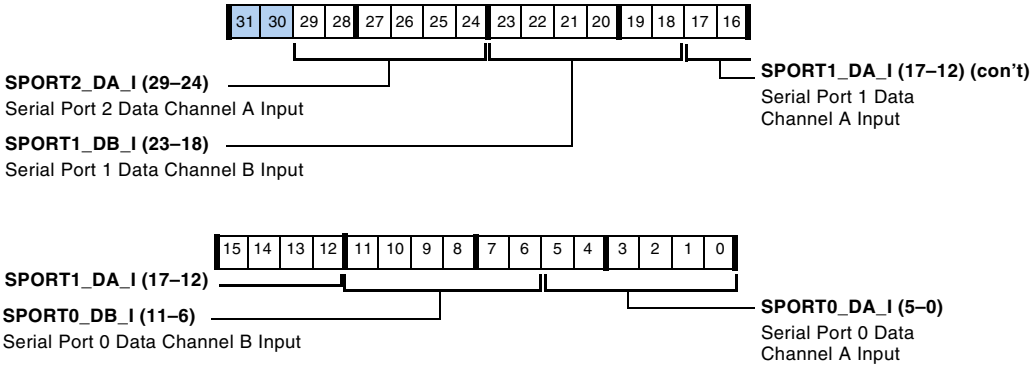

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SPORT2_DA_I (29–24)
Serial Port 2 Data Channel A Input
SPORT1_DB_I (23–18)
Serial Port 1 Data Channel B Input
SPORT1_DA_I (17–12) (con't)
Serial Port 1 Data Channel A Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SPORT1_DA_I (17–12)
SPORT0_DB_I (11–6)
Serial Port 0 Data Channel B Input
SPORT0_DA_I (5–0)
Serial Port 0 Data Channel A Input

Figure A-45. SRU_DAT0 Register

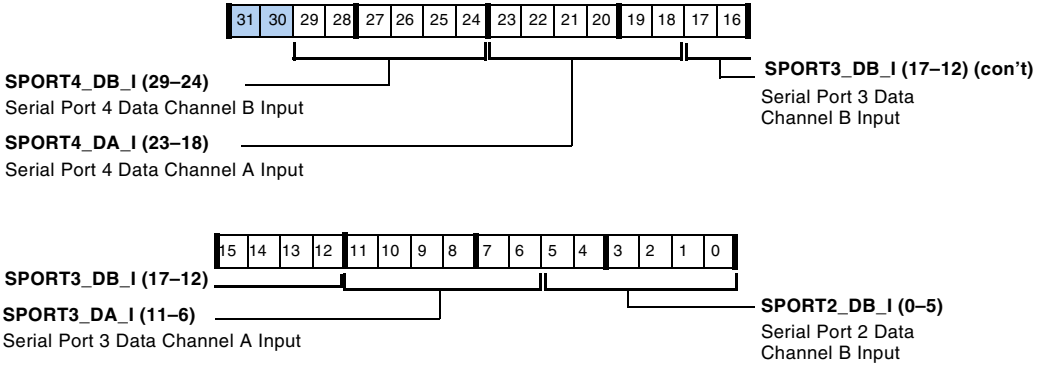

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SPORT4_DB_I (29–24)
Serial Port 4 Data Channel B Input
SPORT4_DA_I (23–18)
Serial Port 4 Data Channel A Input
SPORT3_DB_I (17–12) (con't)
Serial Port 3 Data Channel B Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SPORT3_DB_I (17–12)
SPORT3_DA_I (11–6)
Serial Port 3 Data Channel A Input
SPORT2_DB_I (0–5)
Serial Port 2 Data Channel B Input

Figure A-46. SRU_DAT1 Register

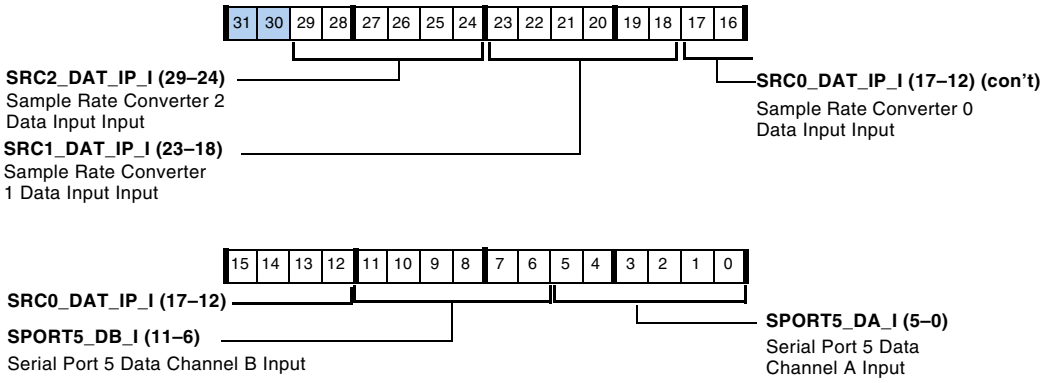

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SRC2_DAT_IP_I (29–24)
Sample Rate Converter 2
Data Input Input
SRC1_DAT_IP_I (23–18)
Sample Rate Converter
1 Data Input Input
SRC0_DAT_IP_I (17–12) (con't)
Sample Rate Converter 0
Data Input Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SRC0_DAT_IP_I (17–12)
SPORT5_DB_I (11–6)
Serial Port 5 Data Channel B Input
SPORT5_DA_I (5–0)
Serial Port 5 Data
Channel A Input

Figure A-47. SRU_DAT2 Register

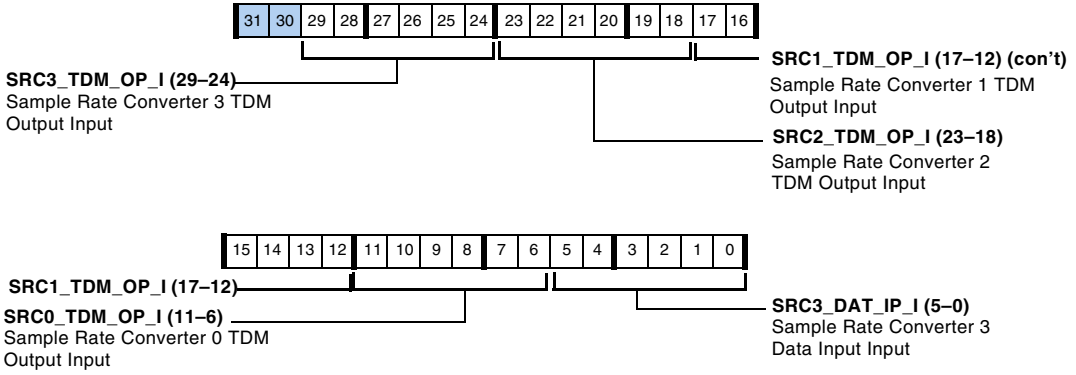

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SRC3_TDM_OP_I (29–24)
Sample Rate Converter 3 TDM
Output Input
SRC1_TDM_OP_I (17–12) (con't)
Sample Rate Converter 1 TDM
Output Input
SRC2_TDM_OP_I (23–18)
Sample Rate Converter 2
TDM Output Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SRC1_TDM_OP_I (17–12)
SRC0_TDM_OP_I (11–6)
Sample Rate Converter 0 TDM
Output Input
SRC3_DAT_IP_I (5–0)
Sample Rate Converter 3
Data Input Input

Figure A-48. SRU_DAT3 Register

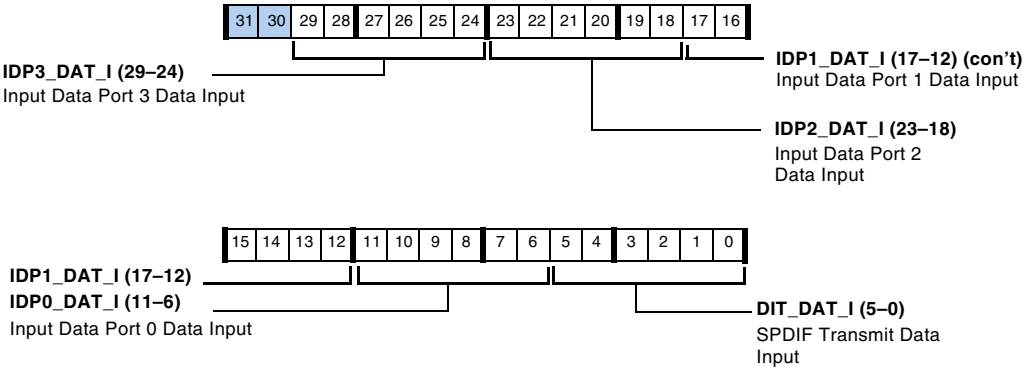

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
IDP3_DAT_I (29–24)
Input Data Port 3 Data Input
IDP1_DAT_I (17–12) (con't)
Input Data Port 1 Data Input
IDP2_DAT_I (23–18)
Input Data Port 2
Data Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
IDP1_DAT_I (17–12)
IDP0_DAT_I (11–6)
Input Data Port 0 Data Input
DIT_DAT_I (5–0)
SPDIF Transmit Data
Input

Figure A-49. SRU_DAT4 Register

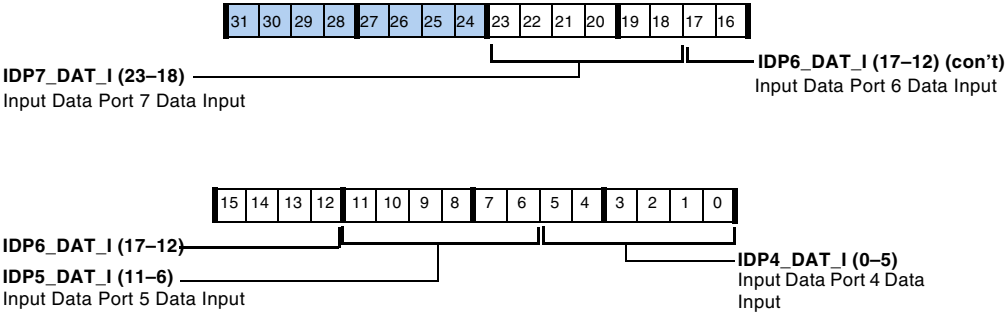

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
IDP7_DAT_I (23–18)
Input Data Port 7 Data Input
IDP6_DAT_I (17–12) (con't)
Input Data Port 6 Data Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
IDP6_DAT_I (17–12)
IDP5_DAT_I (11–6)
Input Data Port 5 Data Input
IDP4_DAT_I (0–5)
Input Data Port 4 Data
Input

Figure A-50. SRU_DAT5 Register

Table A-43. Group B Sources – Serial Data

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	DAI_PB01_O	Pin buffer 1
000001 (0x1)	DAI_PB02_O	Pin buffer 2
000010 (0x2)	DAI_PB03_O	Pin buffer 3
000011 (0x3)	DAI_PB04_O	Pin buffer 4
000100 (0x4)	DAI_PB05_O	Pin buffer 5
000101 (0x5)	DAI_PB06_O	Pin buffer 6

Table A-43. Group B Sources – Serial Data (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
000110 (0x6)	DAI_PB07_O	Pin buffer 7
000111 (0x7)	DAI_PB08_O	Pin buffer 8
001000 (0x8)	DAI_PB09_O	Pin buffer 9
001001 (0x9)	DAI_PB10_O	Pin buffer 10
001010 (0xA)	DAI_PB11_O	Pin buffer 11
001011 (0xB)	DAI_PB12_O	Pin buffer 12
001100 (0xC)	DAI_PB13_O	Pin buffer 13
001101 (0xD)	DAI_PB14_O	Pin buffer 14
001110 (0xE)	DAI_PB15_O	Pin buffer 15
001111 (0xF)	DAI_PB16_O	Pin buffer 16
010000 (0x10)	DAI_PB17_O	Pin buffer 17
010001 (0x11)	DAI_PB18_O	Pin buffer 18
010010 (0x12)	DAI_PB19_O	Pin buffer 19
010011 (0x13)	DAI_PB20_O	Pin buffer 20
010100 (0x14)	SPORT0_DA_O	SPORT 0A data
010101 (0x15)	SPORT0_DB_O	SPORT 0B data
010110 (0x16)	SPORT1_DA_O	SPORT 1A data
010111 (0x17)	SPORT1_DB_O	SPORT 1B data
011000 (0x18)	SPORT2_DA_O	SPORT 2A data
011001 (0x19)	SPORT2_DB_O	SPORT 2B data
011010 (0x1A)	SPORT3_DA_O	SPORT 3A data
011011 (0x1B)	SPORT3_DB_O	SPORT 3B data
011100 (0x1C)	SPORT4_DA_O	SPORT 4A data
011101 (0x1D)	SPORT4_DB_O	SPORT 4B data
011110 (0x1E)	SPORT5_DA_O	SPORT 5A data
011111 (0x1F)	SPORT5_DB_O	SPORT 5B data

Table A-43. Group B Sources – Serial Data (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
100000 (0x20)	SRC0_DAT_OP_O	SRC0 data out
100001 (0x21)	SRC1_DAT_OP_O	SRC1 data out
100010 (0x22)	SRC2_DAT_OP_O	SRC2 data out
100011 (0x23)	SRC3_DAT_OP_O	SRC3 data out
100100 (0x24)	SRC0_TDM_IP_O	SRC0 data out
100101 (0x25)	SRC1_TDM_IP_O	SRC1 data out
100110 (0x26)	SRC2_TDM_IP_O	SRC2 data out
100111 (0x27)	SRC3_TDM_IP_O	SRC3 data out
101000 (0x28)	DIR_DAT_O	SPDIF RX serial data out
101001 (0x29)	LOW	Logic level low (0)
101010 (0x2A)	LOW	Logic level low (0)
101011 (0x2B)	HIGH	Logic level high (1)
101100 (0x2C) – 111111 (0x3F)	Reserved	

Frame Sync Routing Control Registers (SRU_FSx, Group C)

The frame sync routing control registers (see Figure A-51 through Figure A-54) route a frame sync or a word clock to the serial ports, the SRC, the S/PDIF, and the IDP. Each frame sync input is connected to a frame sync source based on the bit values described in the group C frame sync sources, (listed in Table A-44).

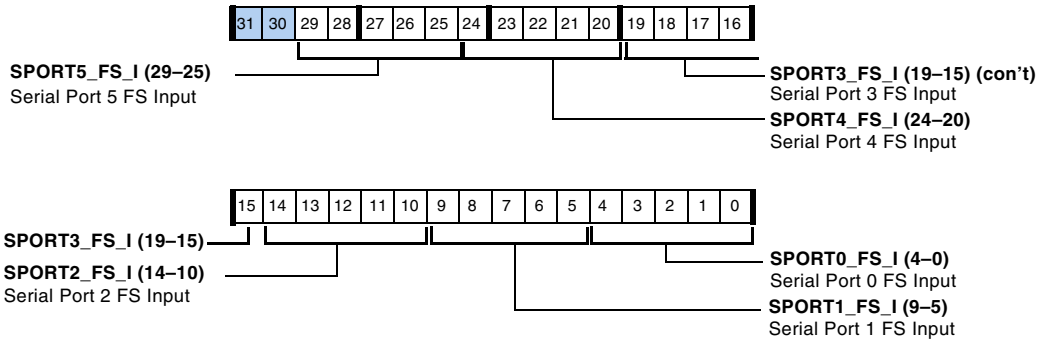

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SPORT5_FS_I (29–25)
Serial Port 5 FS Input
SPORT3_FS_I (19–15) (con't)
Serial Port 3 FS Input
SPORT4_FS_I (24–20)
Serial Port 4 FS Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SPORT3_FS_I (19–15)
SPORT2_FS_I (14–10)
Serial Port 2 FS Input
SPORT0_FS_I (4–0)
Serial Port 0 FS Input
SPORT1_FS_I (9–5)
Serial Port 1 FS Input

Figure A-51. SRU_FS0 Register

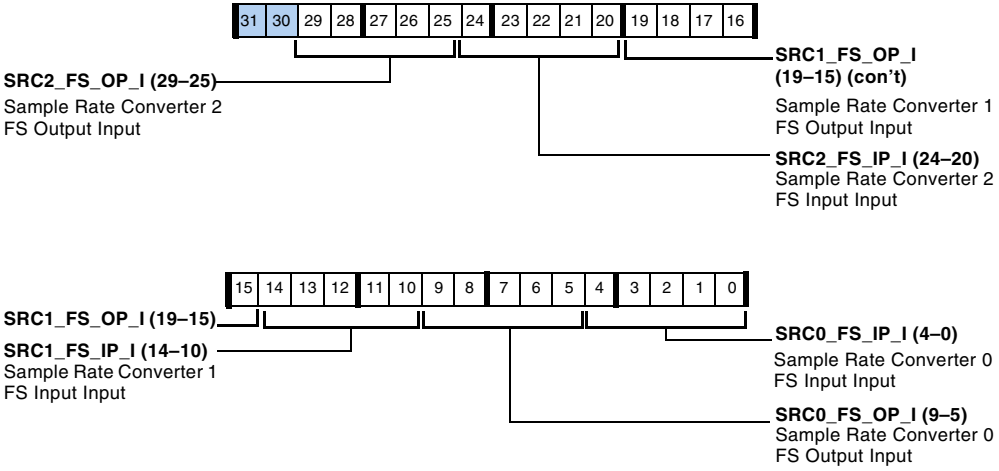

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
SRC2_FS_OP_I (29–25)
Sample Rate Converter 2
FS Output Input
SRC1_FS_OP_I
(19–15) (con't)
Sample Rate Converter 1
FS Output Input
SRC2_FS_IP_I (24–20)
Sample Rate Converter 2
FS Input Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SRC1_FS_OP_I (19–15)
SRC1_FS_IP_I (14–10)
Sample Rate Converter 1
FS Input Input
SRC0_FS_IP_I (4–0)
Sample Rate Converter 0
FS Input Input
SRC0_FS_OP_I (9–5)
Sample Rate Converter 0
FS Output Input

Figure A-52. SRU_FS1 Register

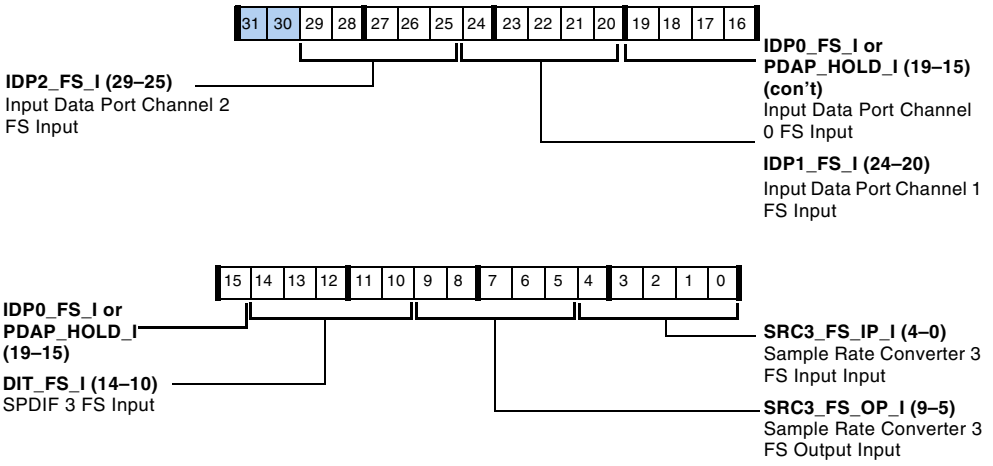


Figure A-53. SRU_FS2 Register

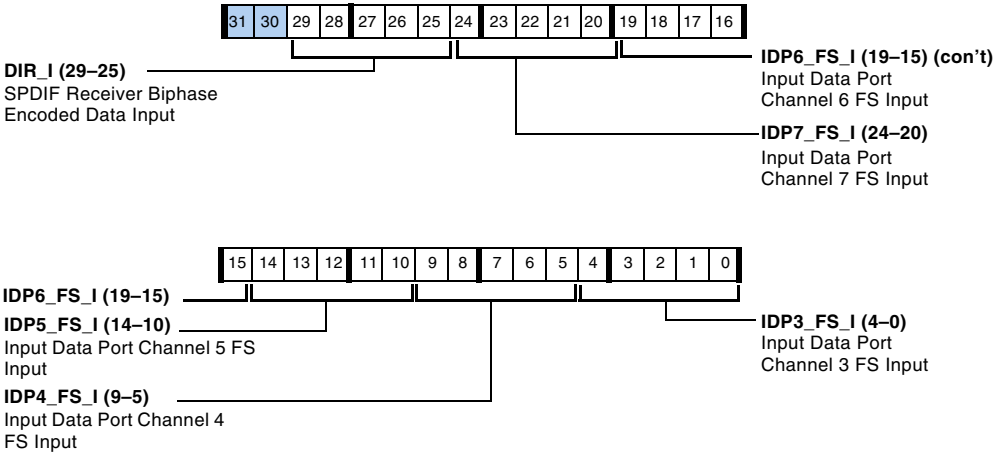

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
DIR_I (29–25)
SPDIF Receiver Biphase
Encoded Data Input
IDP6_FS_I (19–15) (con't)
Input Data Port
Channel 6 FS Input
IDP7_FS_I (24–20)
Input Data Port
Channel 7 FS Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
IDP6_FS_I (19–15)
IDP5_FS_I (14–10)
Input Data Port Channel 5 FS
Input
IDP4_FS_I (9–5)
Input Data Port Channel 4
FS Input
IDP3_FS_I (4–0)
Input Data Port
Channel 3 FS Input

Figure A-54. SRU_FS3 Register

Table A-44. Group C Sources – Frame Sync

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI_PB01_O	Pin buffer 1
00001 (0x1)	DAI_PB02_O	Pin buffer 2
00010 (0x2)	DAI_PB03_O	Pin buffer 3
00011 (0x3)	DAI_PB04_O	Pin buffer 4
00100 (0x4)	DAI_PB05_O	Pin buffer 5
00101 (0x5)	DAI_PB06_O	Pin buffer 6
00110 (0x6)	DAI_PB07_O	Pin buffer 7
00111 (0x7)	DAI_PB08_O	Pin buffer 8
01000 (0x8)	DAI_PB09_O	Pin buffer 9
01001 (0x9)	DAI_PB10_O	Pin buffer 10
01010 (0xA)	DAI_PB11_O	Pin buffer 11
01011 (0xB)	DAI_PB12_O	Pin buffer 12
01100 (0xC)	DAI_PB13_O	Pin buffer 13
01101 (0xD)	DAI_PB14_O	Pin buffer 14
01110 (0xE)	DAI_PB15_O	Pin buffer 15
01111 (0xF)	DAI_PB16_O	Pin buffer 16
10000 (0x10)	DAI_PB17_O	Pin buffer 17
10001 (0x11)	DAI_PB18_O	Pin buffer 18
10010 (0x12)	DAI_PB19_O	Pin buffer 19
10011 (0x13)	DAI_PB20_O	Pin buffer 20
10100 (0x14)	SPORT0_FS_O	SPORT 0 frame sync
10101 (0x15)	SPORT1_FS_O	SPORT 1 frame sync
10110 (0x16)	SPORT2_FS_O	SPORT 2 frame sync
10111 (0x17)	SPORT3_FS_O	SPORT 3 frame sync
11000 (0x18)	SPORT4_FS_O	SPORT 4 frame sync
11001 (0x19)	SPORT5_FS_O	SPORT 5 frame sync

Table A-44. Group C Sources – Frame Sync (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
11010 (0x1A)	DIR_FS_O	SPDIF RX frame sync output
11011 (0x1B)	DIT_O	SPDIF TX biphase encoded output
11100 (0x1C)	PCG_FSA_O	Precision frame sync A output
11101 (0x1D)	PCG_FSB_O	Precision frame sync B output
11110 (0x1E)	LOW	Logic level low (0)
11111 (0x1F)	HIGH	Logic level high (1)

Pin Signal Assignment Registers (SRU_PINx, Group D)

Each physical pin (connected to a bonded pad) may be routed using the pin signal assignment registers (see Figure A-55 through Figure A-59) in the SRU to any of the inputs or outputs of the DAI peripherals, based on the 7-bit values listed in Table A-45. The SRU also may be used to route signals that control the pins in other ways.

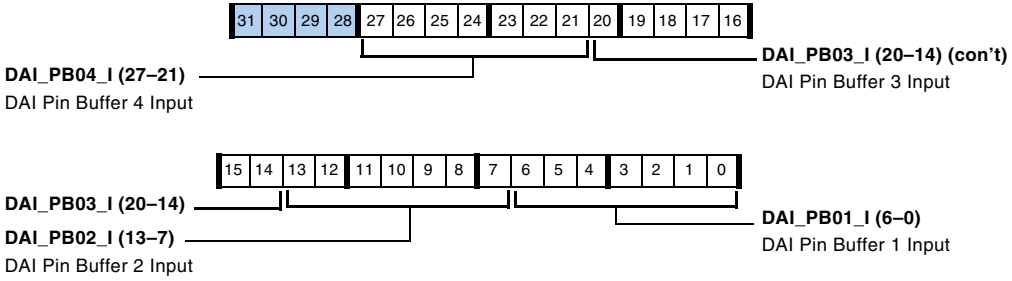

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
DAI_PB04_I (27–21)
DAI Pin Buffer 4 Input
DAI_PB03_I (20–14) (con't)
DAI Pin Buffer 3 Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DAI_PB03_I (20–14)
DAI_PB02_I (13–7)
DAI Pin Buffer 2 Input
DAI_PB01_I (6–0)
DAI Pin Buffer 1 Input

Figure A-55. SRU_PIN0 Register

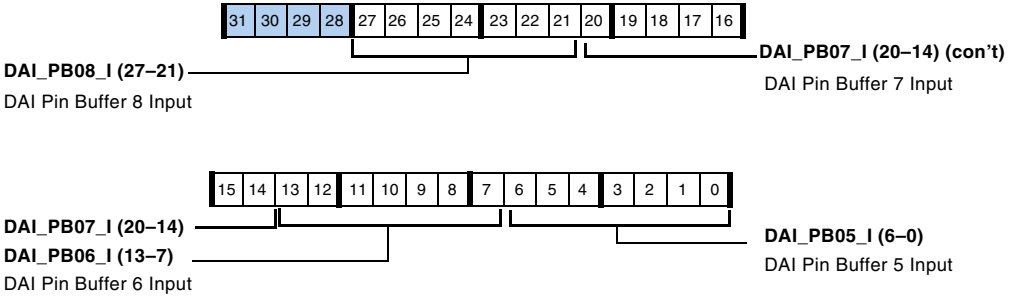

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
DAI_PB08_I (27–21)
DAI Pin Buffer 8 Input
DAI_PB07_I (20–14) (con't)
DAI Pin Buffer 7 Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DAI_PB07_I (20–14)
DAI_PB06_I (13–7)
DAI Pin Buffer 6 Input
DAI_PB05_I (6–0)
DAI Pin Buffer 5 Input

Figure A-56. SRU_PIN1 Register

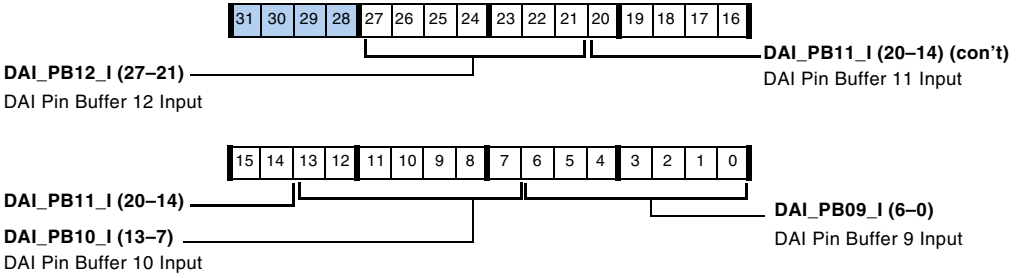

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
DAI_PB12_I (27–21)
DAI Pin Buffer 12 Input
DAI_PB11_I (20–14) (con't)
DAI Pin Buffer 11 Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DAI_PB11_I (20–14)
DAI_PB10_I (13–7)
DAI Pin Buffer 10 Input
DAI_PB09_I (6–0)
DAI Pin Buffer 9 Input

Figure A-57. SRU_PIN2 Register

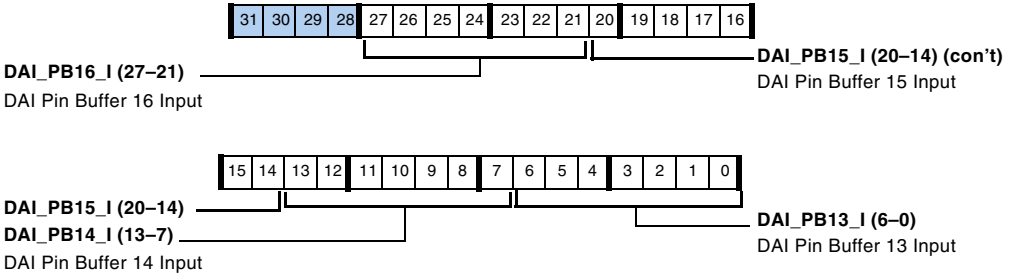

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
DAI_PB16_I (27–21)
DAI Pin Buffer 16 Input
DAI_PB15_I (20–14) (con't)
DAI Pin Buffer 15 Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DAI_PB15_I (20–14)
DAI_PB14_I (13–7)
DAI Pin Buffer 14 Input
DAI_PB13_I (6–0)
DAI Pin Buffer 13 Input

Figure A-58. SRU_PIN3 Register

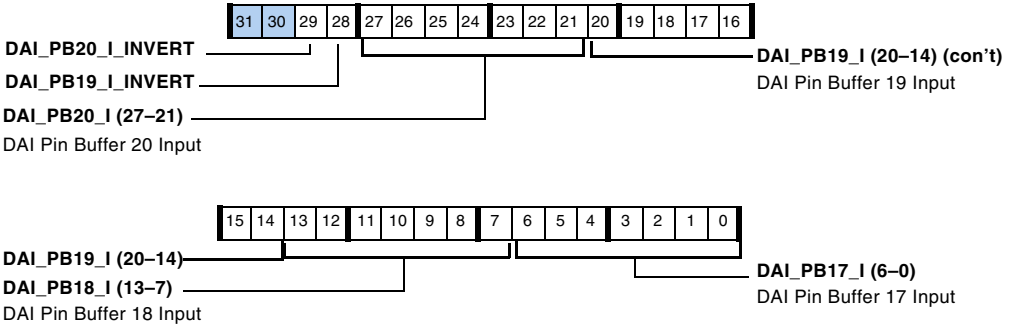

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
DAI_PB20_I_INVERT
DAI_PB19_I_INVERT
DAI_PB20_I (27–21)
DAI Pin Buffer 20 Input
DAI_PB19_I (20–14) (con't)
DAI Pin Buffer 19 Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DAI_PB19_I (20–14)
DAI_PB18_I (13–7)
DAI Pin Buffer 18 Input
DAI_PB17_I (6–0)
DAI Pin Buffer 17 Input

Figure A-59. SRU_PIN4 Register

Setting SRU_PIN4 bit 28 to high inverts the level of DAI_PB19_I and setting SRU_PIN4 bit 29 to high inverts the level of DAI_PB20_I. Input Inversion only works if the buffer output is not routed to its input.

Table A-45. Group D Sources – Pin Signal Assignments

Selection Code	Source Signal	Description (Source Selection)
0000000 (0x0)	DAI_PB01_O	Pin buffer 1
0000001 (0x1)	DAI_PB02_O	Pin buffer 2
0000010 (0x2)	DAI_PB03_O	Pin buffer 3
0000011 (0x3)	DAI_PB04_O	Pin buffer 4
0000100 (0x4)	DAI_PB05_O	Pin buffer 5
0000101 (0x5)	DAI_PB06_O	Pin buffer 6
0000110 (0x6)	DAI_PB07_O	Pin buffer 7
0000111 (0x7)	DAI_PB08_O	Pin buffer 8
0001000 (0x8)	DAI_PB09_O	Pin buffer 9
0001001 (0x9)	DAI_PB10_O	Pin buffer 10
0001010 (0xA)	DAI_PB11_O	Pin buffer 11
0001011 (0xB)	DAI_PB12_O	Pin buffer 12
0001100 (0xC)	DAI_PB13_O	Pin buffer 13

Table A-45. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
0001101 (0xD)	DAI_PB14_O	Pin buffer 14
0001110 (0xE)	DAI_PB15_O	Pin buffer 15
0001111 (0xF)	DAI_PB16_O	Pin buffer 16
0010000 (0x10)	DAI_PB17_O	Pin buffer 17
0010001 (0x11)	DAI_PB18_O	Pin buffer 18
0010010 (0x12)	DAI_PB19_O	Pin buffer 19
0010011 (0x13)	DAI_PB20_O	Pin buffer 20
0010100 (0x14)	SPORT0_DA_O	SPORT 0A data
0010101 (0x15)	SPORT0_DB_O	SPORT 0B data
0010110 (0x16)	SPORT1_DA_O	SPORT 1A data
0010111 (0x17)	SPORT1_DB_O	SPORT 1B data
0011000 (0x18)	SPORT2_DA_O	SPORT 2A data
0011001 (0x19)	SPORT2_DB_O	SPORT 2B data
0011010 (0x1A)	SPORT3_DA_O	SPORT 3A data
0011011 (0x1B)	SPORT3_DB_O	SPORT 3B data
0011100 (0x1C)	SPORT4_DA_O	SPORT 4A data
0011101 (0x1D)	SPORT4_DB_O	SPORT 4B data
0011110 (0x1E)	SPORT5_DA_O	SPORT 5A data
0011111 (0x1F)	SPORT5_DB_O	SPORT 5B data
0100000 (0x20)	SPORT0_CLK_O	SPORT 0 clock
0100001 (0x21)	SPORT1_CLK_O	SPORT 1 clock
0100010 (0x22)	SPORT2_CLK_O	SPORT 2 clock
0100011 (0x23)	SPORT3_CLK_O	SPORT 3 clock
0100100 (0x24)	SPORT4_CLK_O	SPORT 4 clock
0100101 (0x25)	SPORT5_CLK_O	SPORT 5 clock
0100110 (0x26)	SPORT0_FS_O	SPORT 0 frame sync

Table A-45. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
0100111 (0x27)	SPORT1_FS_O	SPORT 1 frame sync
0101000 (0x28)	SPORT2_FS_O	SPORT 2 frame sync
0101001 (0x29)	SPORT3_FS_O	SPORT 3 frame sync
0101010 (0x2A)	SPORT4_FS_O	SPORT 4 frame sync
0101011 (0x2B)	SPORT5_FS_O	SPORT 5 frame sync
0101100 (0x2C)	TIMER0_O	timer 0
0101101 (0x2D)	TIMER1_O	timer 1
0101110 (0x2E)	TIMER2_O	timer 2
0101111 (0x2F)	FLAG10_O	flag 10
0110000 (0x30)	PDAP_STRB_O	PDAP data transfer request strobe
0110000 (0x31)– 0110011 (0x33)	Reserved	
0110100 (0x34)	FLAG11_O	flag 11
00110101 (0x35)	FLAG12_O	flag 12
0110110 (0x36)	FLAG13_O	flag 13
0110111 (0x37)	FLAG14_O	flag 14
0111000 (0x38)	PCG_CLKA_O	precision clock A
0111001 (0x39)	PCG_CLKB_O	precision clock B
0111010 (0x3A)	PCG_FSA_O	precision frame sync A
0111011 (0x3B)	PCG_FSB_O	precision frame sync B
1111000 (0x3C)	FLAG15_O	flag 15
0111101 (0x3D)	SRC0_DAT_OP_O	SRC0 data output
0111110 (0x3E)	SRC1_DAT_OP_O	SRC1 data output
0111111 (0x3F)	SRC2_DAT_OP_O	SRC2 data output
1000000 (0x40)	SRC3_DAT_OP_O	SRC3 data output
1000001 (0x41)	DIR_DAT_O	SPDIF RX data output

Table A-45. Group D Sources – Pin Signal Assignments (Cont'd)

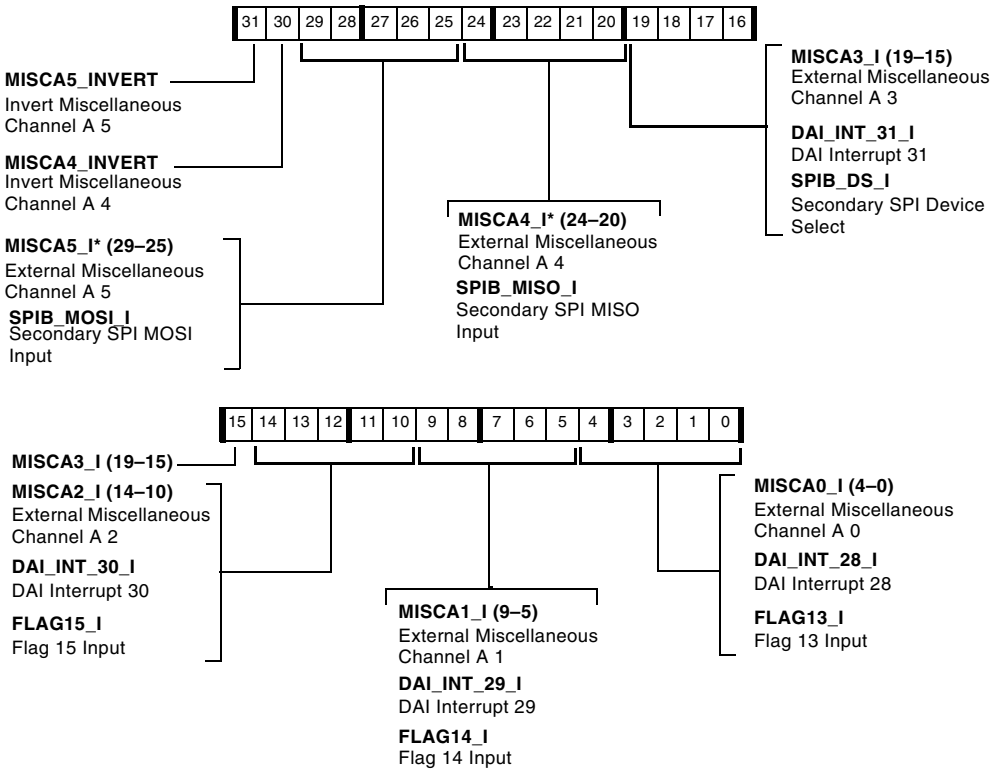
Selection Code	Source Signal	Description (Source Selection)
1000010 (0x42)	DIR_FS_O	SPDIF RX frame sync output
1000011 (0x43)	DIR_CLK_O	SPDIF RX clock output
1000100 (0x44)	DIR_TDMCLK_O	SPDIF RX TDM clock output
1000101 (0x45)	DIT_O	SPDIF TX biphase encoded output
1000110 (0x46)	SPIB_MISO_O	MISO from SPIB
1000111 (0x47)	SPIB_MOSI_O	MOSI from SPIB
1001000 (0x48)	SPIB_CLK_O	Clock from SPIB
1001001 (0x49)	SPIB_FLG0_O	Slave 0 from SPIB
1001010 (0x4A)	SPIB_FLG1_O	Slave 1 from SPIB
1001011 (0x4B)	SPIB_FLG2_O	Slave 2 from SPIB
1001100 (0x4C)	SPIB_FLG3_O	Slave 3 from SPIB
1001101 (0x4D)	DIR_LRCLK_FB_O	External PLL – feedback point connection
1001110 (0x4E)	DIR_LRCLK_REF_O	External PLL – reference point connection
1001111 (0x4F)	Reserved	
1010000 (0x50)	LOW	Logic level low (0)
1010001 (0x51)	HIGH	Logic level high (1)
1010010 – 1111111 (0x52 – 0x7F)	Reserved	

Miscellaneous Signal Routing Registers (SRU_MISCx, Group E)

The miscellaneous registers (see Figure A-60 and Figure A-61) are a very powerful and versatile feature of the DAI. These registers allow external pins, timers, and clocks to serve as interrupt sources or timer inputs and outputs. They also allow pins to connect to other pins, or to invert the logic of other pins. Notice that when the PCG's one shot option (PCG_PW

register) is enabled, the inputs MISCA2_I (PCG unit A) and MISCA3_I (PCG unit B) are used asinput signals. Also notice if using the S/PDIF Tx block start output, it must be routed to the MISCB4_I input for interrupt operation.

The miscellaneous signal routing registers correspond to the group E miscellaneous signals, listed in Table A-46.



***Setting SRU_MISCA[30] to high inverts the level of MISCA4_I, and setting SRU_MISCA[31] to high inverts the level of MISCA5_I.**

Figure A-60. SRU_EXT_MISCA Register

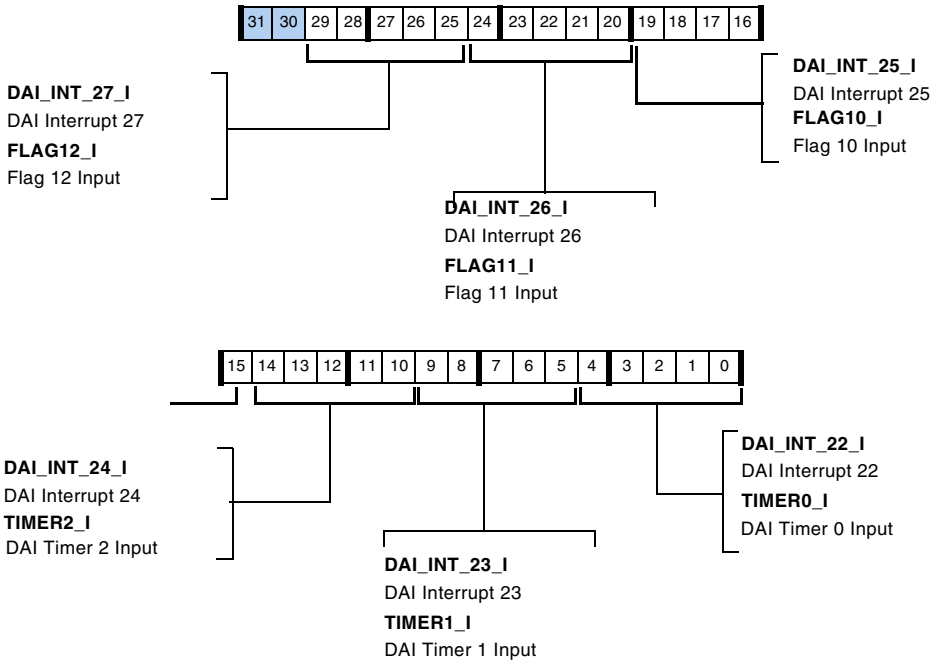


Figure A-61. SRU_EXT_MISCB Register

Table A-46. Group E Sources – Miscellaneous Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI_PB01_O	Pin buffer 1 output as a source
00001 (0x1)	DAI_PB02_O	Pin buffer 2 output as a source
00010 (0x2)	DAI_PB03_O	Pin buffer 3 output as a source
00011 (0x3)	DAI_PB04_O	Pin buffer 4 output as a source
00100 (0x4)	DAI_PB05_O	Pin buffer 5 output as a source
00101 (0x5)	DAI_PB06_O	Pin buffer 6 output as a source
00110 (0x6)	DAI_PB07_O	Pin buffer 7 output as a source
00111 (0x7)	DAI_PB08_O	Pin buffer 8 output as a source
01000 (0x8)	DAI_PB09_O	Pin buffer 9 output as a source

Table A-46. Group E Sources – Miscellaneous Signals (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
01001 (0x9)	DAI_PB10_O	Pin buffer 10 output as a source
01010 (0xA)	DAI_PB11_O	Pin buffer 11 output as a source
01011 (0xB)	DAI_PB12_O	Pin buffer 12 output as a source
01100 (0xC)	DAI_PB13_O	Pin buffer 13 output as a source
01101 (0xD)	DAI_PB14_O	Pin buffer 14 output as a source
01110 (0xE)	DAI_PB15_O	Pin buffer 15 output as a source
01111 (0xF)	DAI_PB16_O	Pin buffer 16 output as a source
10000 (0x10)	DAI_PB17_O	Pin buffer 17 output as a source
10001 (0x11)	DAI_PB18_O	Pin buffer 18 output as a source
10010 (0x12)	DAI_PB19_O	Pin buffer 19 output as a source
10011 (0x13)	DAI_PB20_O	Pin buffer 20 output as a source
10100 (0x14)	TIMER0_O	GP Timer 0 output as a source
10101 (0x15)	TIMER1_O	GP Timer 1 output as a source
10110 (0x16)	TIMER2_O	GP Timer 2 output as a source
10110 (0x17)	Reserved	
11000 (0x18)	DIT_BLKSTART_O	SPDIF Transmit Block start signal as a source
11001 (0x19)	PDAP_STRB_O	PDAP strobe output as a source
11010 (0x1A)	PCG_CLKA_O	Precision clock A output as a source
11011 (0x1B)	PCG_FSA_O	Precision frame sync A output as a source
11100 (0x1C)	PCG_CLKB_O	Precision clock B output as a source
11101 (0x1D)	PCG_FSB_O	Precision frame sync B output as a source
11110 (0x1E)	LOW	Logic level low (0)
11111 (0x1F)	HIGH	Logic level high (1)

DAI Pin Buffer Enable Registers (SRU_PBENx, Group F)

The pin enable control registers (see Figure A-62 through Figure A-65) activate the drive buffer for each of the 20 DAI pins. When the pins are not enabled (driven), they can be used as inputs. Each of the pin enables are connected, based on the 6-bit values in Table A-47. Notice all SPORTx_x_PBEN_O signals only driven if the serial ports are in master mode.

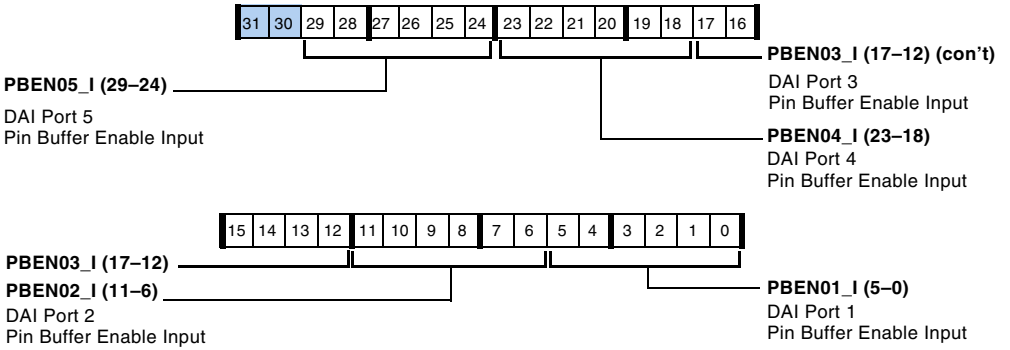

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
PBEN05_I (29–24)
DAI Port 5
Pin Buffer Enable Input
PBEN03_I (17–12) (con't)
DAI Port 3
Pin Buffer Enable Input
PBEN04_I (23–18)
DAI Port 4
Pin Buffer Enable Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
PBEN03_I (17–12)
PBEN02_I (11–6)
DAI Port 2
Pin Buffer Enable Input
PBEN01_I (5–0)
DAI Port 1
Pin Buffer Enable Input

Figure A-62. SRU_PBEN0

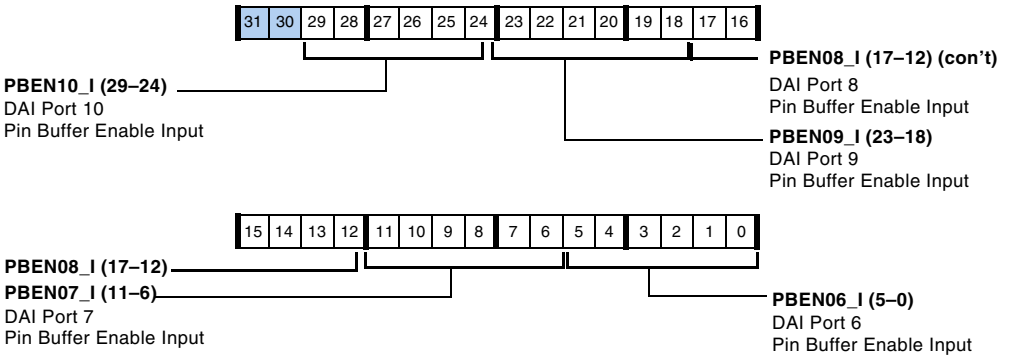

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
PBEN10_I (29–24)
DAI Port 10
Pin Buffer Enable Input
PBEN08_I (17–12) (con't)
DAI Port 8
Pin Buffer Enable Input
PBEN09_I (23–18)
DAI Port 9
Pin Buffer Enable Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
PBEN08_I (17–12)
PBEN07_I (11–6)
DAI Port 7
Pin Buffer Enable Input
PBEN06_I (5–0)
DAI Port 6
Pin Buffer Enable Input

Figure A-63. SRU_PBEN1

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
PBEN15_I (29–24)
DAI Port 15
Pin Buffer Enable Input
PBEN13_I (17–12) (con't)
PBEN14_I (23–18)
DAI Port 14
Pin Buffer Enable Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
PBEN13_I (17–12)
DAI Port 13
Pin Buffer Enable Input
PBEN11_I (5–0)
DAI Port 11
Pin Buffer Enable Input
PBEN12_I (11–6)
DAI Port 12
Pin Buffer Enable Input

Figure A-64. SRU_PBEN2

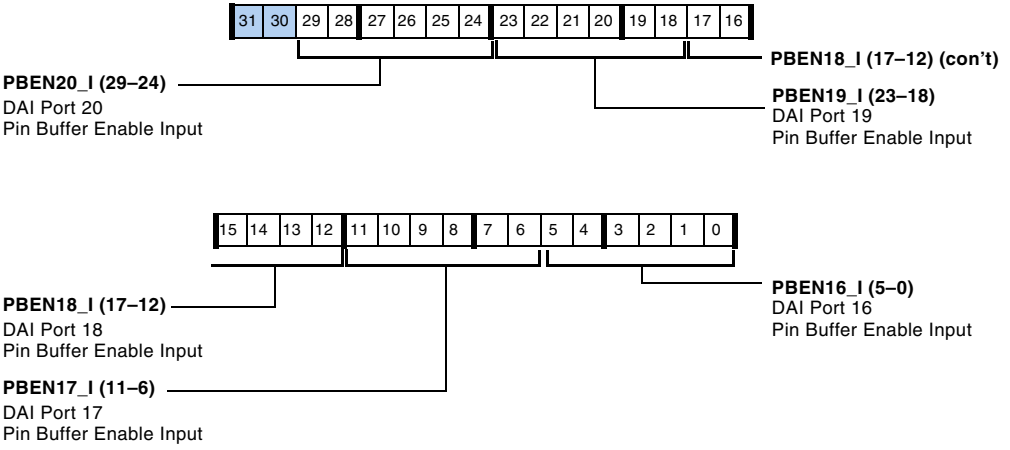

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
PBEN20_I (29–24)
DAI Port 20
Pin Buffer Enable Input
PBEN18_I (17–12) (con't)
PBEN19_I (23–18)
DAI Port 19
Pin Buffer Enable Input
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
PBEN18_I (17–12)
DAI Port 18
Pin Buffer Enable Input
PBEN16_I (5–0)
DAI Port 16
Pin Buffer Enable Input
PBEN17_I (11–6)
DAI Port 17
Pin Buffer Enable Input

Figure A-65. SRU_PBEN3

Table A-47. Group F Sources – Pin Output Enable

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	LOW	Logic level low (0)
000001 (0x1)	HIGH	Logic level high (1)
000010 (0x2)	MISCA0_O	Miscellaneous control A0 output
000011 (0x3)	MISCA1_O	Miscellaneous control A1 output
000100 (0x4)	MISCA2_O	Miscellaneous control A2 output
000101 (0x5)	MISCA3_O	Miscellaneous control A3 output
000110 (0x6)	MISCA4_O	Miscellaneous control A4 output
000111 (0x7)	MISCA5_O	Miscellaneous control A5 output
001000 (0x8)	SPORT0_CLK_PBEN_O	SPORT 0 clock output enable
001001 (0x9)	SPORT0_FS_PBEN_O	SPORT 0 frame sync output enable
001010 (0xA)	SPORT0_DA_PBEN_O	SPORT 0 data channel A output enable
001011 (0xB)	SPORT0_DB_PBEN_O	SPORT 0 data channel B output enable
001100 (0xC)	SPORT1_CLK_PBEN_O	SPORT 1 clock output enable
001101 (0xD)	SPORT1_FS_PBEN_O	SPORT 1 frame sync output enable
001110 (0xE)	SPORT1_DA_PBEN_O	SPORT 1 data channel A output enable
001111 (0xF)	SPORT1_DB_PBEN_O	SPORT 1 data channel B output enable
010000 (0x10)	SPORT2_CLK_PBEN_O	SPORT 2 clock output enable
010001 (0x11)	SPORT2_FS_PBEN_O	SPORT 2 frame sync output enable
010010 (0x12)	SPORT2_DA_PBEN_O	SPORT 2 data channel A output enable
010011 (0x13)	SPORT2_DB_PBEN_O	SPORT 2 data channel B output enable
010100 (0x14)	SPORT3_CLK_PBEN_O	SPORT 3 clock output enable
010101 (0x15)	SPORT3_FS_PBEN_O	SPORT 3 frame sync output enable
010110 (0x16)	SPORT3_DA_PBEN_O	SPORT 3 data channel A output enable
010111 (0x17)	SPORT3_DB_PBEN_O	SPORT 3 data channel B output enable
011000 (0x18)	SPORT4_CLK_PBEN_O	SPORT 4 clock output enable
011001 (0x19)	SPORT4_FS_PBEN_O	SPORT 4 frame sync output enable

Table A-47. Group F Sources – Pin Output Enable (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
011010 (0x1A)	SPORT4_DA_PBEN_O	SPORT 4 data channel A output enable
011011 (0x1B)	SPORT4_DB_PBEN_O	SPORT 4 data channel B output enable
011100 (0x1C)	SPORT5_CLK_PBEN_O	SPORT 5 clock as the output enable source
011101 (0x1D)	SPORT5_FS_PBEN_O	SPORT 5 frame sync output enable
011110 (0x1E)	SPORT5_DA_PBEN_O	SPORT 5 data channel A output enable
011111 (0x1F)	SPORT5_DB_PBEN_O	SPORT 5 data channel B output enable
100000 (0x20)	TIMER0_PBEN_O	Timer 0 output enable
100001 (0x21)	TIMER1_PBEN_O	Timer 1 output enable
100010 (0x22)	TIMER2_PBEN_O	Timer 2 output enable
100011 (0x23)	FLAG10_PBEN_O	Flag 10 output enable
100100 (0x24)	FLAG11_PBEN_O	Flag 11 output enable
100101 (0x25)	FLAG12_PBEN_O	Flag 12 output enable
100110 (0x26)	FLAG13_PBEN_O	Flag 13 output enable
100111 (0x27)	FLAG14_PBEN_O	Flag 14 output enable
101000 (0x28)	FLAG15_PBEN_O	Flag 15 output enable
101001 (0x29)	SPIB_MISO_PBEN_O	Pin enable for MISO from SPIB
101010 (0x2A)	SPIB_MOSI_PBEN_O	Pin enable for MOSI from SPIB
101011 (0x2B)	SPIB_CLK_PBEN_O	Pin enable for clock from SPIB
101100 (0x2C)	SPIB_FLG0_PBEN_O	Pin enable for slave 0 from SPIB
101101 (0x2D)	SPIB_FLG1_PBEN_O	Pin enable for slave 1 from SPIB
100111 (0x2E)	SPIB_FLG2_PBEN_O	Pin enable for slave 2 from SPIB
101110 (0x2F)	SPIB_FLG3_PBEN_O	Pin enable for slave 3 from SPIB
1100000 (0x30)– 1111111 (0x3F)	Reserved	

DAI Status Registers

The DAI_STAT register, shown in Figure A-39 and described in Table A-41, and the DAI_PIN_STAT register, shown in Figure A-67 on page A-107, provide status information for the IDP/PDAP DMA channels.

DAI Pin Buffer Registers (DAI_PIN_PULLUP, DAI_PIN_STAT)

The DAI_PIN_STAT register, shown in Figure A-67, provides DAI pin buffer level status information. Reading 0 indicates low level of V_{DDEXT}, reading a 1 indicates a high level of V_{DDEXT}. information. The DAI_PIN_PULLUP register, shown in Figure A-66, allows programs to enable/disable pull-up resistors. (Bits 19–0 of this register control the pull-up resistors on DAI_P20–1.) Setting a bit to 1 enables a pull-up resistor on the corresponding pin. After reset, the pull-up resistors are enabled on all 20 DAI pins.

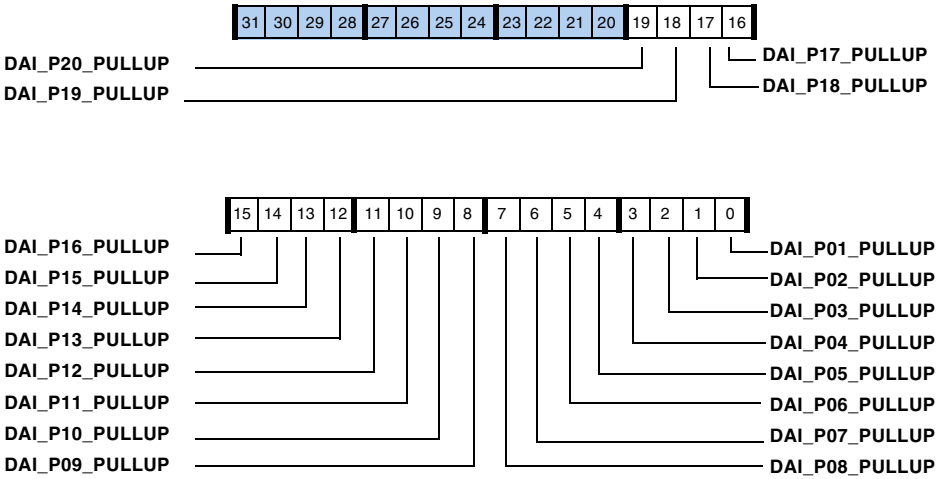


Figure A-66. DAI_PIN_PULLUP Register

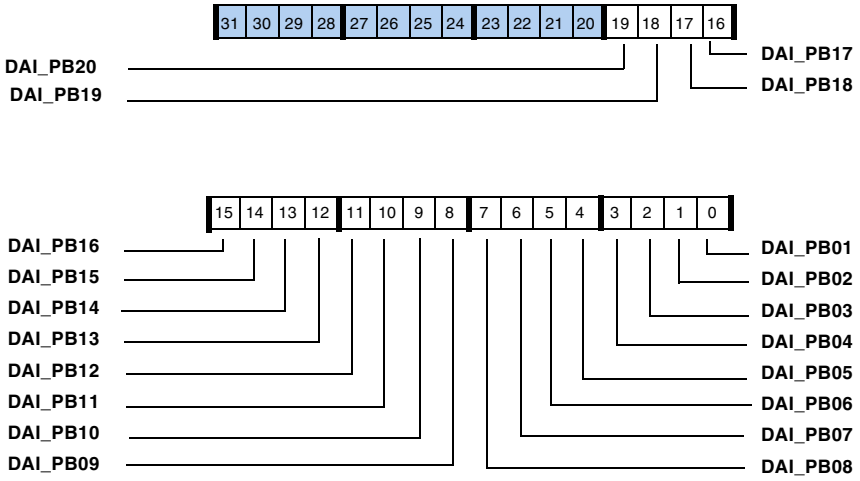


Figure A-67. DAI_PIN_STAT Register

Register Listing

This section list all available memory mapped IOP registers including the address and reset values. For core register listings, see the *SHARC Processor Programming Reference*.

Register Mnemonic	Address	Description	Reset
Serial Port 0 and 1 Registers			
SPCTL0	0xC00	SPORT 0 Control Register	0x0000 0000
SPCTL1	0xC01	SPORT 1 Control Register	0x0000 0000
DIV0	0xC02	SPORT 0 divisor for TX/RX SCLK0 and FS0	0x0
DIV1	0xC03	SPORT 1 divisor for TX/RX SCLK1 and FS1	0x0
SPMCTL01	0xC04	SPORTs 0 and 1 Multichannel Control Register	0x0
MT0CS0	0xC05	SPORT 0 multichannel TX select, channels 31–0	Undefined
MT0CS1	0xC06	SPORT 0 multichannel TX select, channels 63–32	Undefined
MT0CS2	0xC07	SPORT 0 multichannel TX select, channels 95–64	Undefined
MT0CS3	0xC08	SPORT 0 multichannel TX select, channels 127–96	Undefined
MR1CS0	0xC09	SPORT 1 multichannel RX select, channels 31–0	Undefined
MR1CS1	0xC0A	SPORT 1 multichannel RX select, channels 63–32	Undefined
MR1CS2	0xC0B	SPORT 1 multichannel RX select, channels 95–64	Undefined
MR1CS3	0xC0C	SPORT 1 multichannel RX select, channels 127–96	Undefined
MT0CCS0	0xC0D	SPORT 0 multichannel TX compand select, channels 31–0	Undefined
MT0CCS1	0xC0E	SPORT 0 multichannel TX compand select, channels 63–32	Undefined
MT0CCS2	0xC0F	SPORT 0 multichannel TX compand select, channels 95–64	Undefined
MT0CCS3	0xC10	SPORT 0 multichannel TX compand select, channels 127–96	Undefined
MR1CCS0	0xC11	SPORT 1 multichannel RX compand select, channels 31–0	Undefined
MR1CCS1	0xC12	SPORT 1 multichannel RX compand select, channels 63–32	Undefined
MR1CCS2	0xC13	SPORT 1 multichannel RX compand select, channels 95–64	Undefined
MR1CCS3	0xC14	SPORT 1 multichannel RX compand select, channels 127–96	Undefined

Register Mnemonic	Address	Description	Reset
IISP0A	0xC40	Internal memory DMA address	Undefined
IMSP0A	0xC41	Internal memory DMA access modifier	Undefined
CSP0A	0xC42	Contains number of DMA transfers remaining	Undefined
CPSP0A	0xC43	Points to next DMA parameters	Undefined
IISP0B	0xC44	Internal memory DMA address	Undefined
IMSP0B	0xC45	Internal memory DMA access modifier	Undefined
CSP0B	0xC46	Contains number of DMA transfers remaining	Undefined
CPSP0B	0xC47	Points to next DMA parameters	Undefined
IISP1A	0xC48	Internal memory DMA address	Undefined
IMSP1A	0xC49	Internal memory DMA access modifier	Undefined
CSP1A	0xC4A	Contains number of DMA transfers remaining	Undefined
CPSP1A	0xC4B	Points to next DMA parameters	Undefined
IISP1B	0xC4C	Internal memory DMA address	Undefined
IMSP1B	0xC4D	Internal memory DMA access modifier	Undefined
CSP1B	0xC4E	Contains number of DMA transfers remaining	Undefined
CPSP1B	0xC4F	Points to next DMA parameters	Undefined
TXSP0A	0xC60	SPORT 0A transmit data	Undefined
RXSP0A	0xC61	SPORT 0A receive data	Undefined
TXSP0B	0xC62	SPORT 0B transmit data	Undefined
RXSP0B	0xC63	SPORT 0B receive data	Undefined
TXSP1A	0xC64	SPORT 1A transmit data	Undefined
RXSP1A	0xC65	SPORT 1A receive data	Undefined
TXSP1B	0xC66	SPORT 1B transmit data	Undefined
RXSP1B	0xC67	SPORT 1B receive data	Undefined
Serial Port 2 and 3 Registers			
SPCTL2	0x400	SPORT 2 control	0x0000 0000
SPCTL3	0x401	SPORT 3 control	0x0000 0000

Register Mnemonic	Address	Description	Reset
DIV2	0x402	SPORT 2 divisor for TX/RX SCLK2 and FS2	0x0
DIV3	0x403	SPORT 3 divisor for TX/RX SCLK3 and FS3	0x0
SPMCTL23	0x404	SPORTs 2 & 3 Multichannel Control	0x0
MT2CS0	0x405	SPORT 2 multichannel TX select, channels 31–0	Undefined
MT2CS1	0x406	SPORT 2 multichannel TX select, channels 63–32	Undefined
MT2CS2	0x407	SPORT 2 multichannel TX select, channels 95–64	Undefined
MT2CS3	0x408	SPORT 2 multichannel TX select, channels 127–96	Undefined
MR3CS0	0x409	SPORT 3 multichannel RX select, channels 31–0	Undefined
MR3CS1	0x40A	SPORT 3 multichannel RX select, channels 63–32	Undefined
MR3CS2	0x40B	SPORT 3 multichannel RX select, channels 95–64	Undefined
MR3CS3	0x40C	SPORT 3 multichannel RX select, channels 127–96	Undefined
MT2CCS0	0x40D	SPORT 2 multichannel TX compand select, channels 31–0	Undefined
MT2CCS1	0x40E	SPORT 2 multichannel TX compand select, channels 63–32	Undefined
MT2CCS2	0x40F	SPORT 2 multichannel TX compand select, channels 95–64	Undefined
MT2CCS3	0x410	SPORT 2 multichannel TX compand select, channels 127–96	Undefined
MR3CCS0	0x411	SPORT 3 multichannel RX compand select, channels 31–0	Undefined
MR3CCS1	0x412	SPORT 3 multichannel RX compand select, channels 63–32	Undefined
MR3CCS2	0x413	SPORT 3 multichannel RX compand select, channels 95–64	Undefined
MR3CCS3	0x414	SPORT 3 multichannel RX compand select, channels 127–96	Undefined
IISP2A	0x440	Internal memory DMA address	Undefined
IMSP2A	0x441	Internal memory DMA access modifier	Undefined
CSP2A	0x442	Contains number of DMA transfers remaining	Undefined
CPSP2A	0x443	Points to next DMA parameters	Undefined
IISP2B	0x444	Internal memory DMA address	Undefined
IMSP2B	0x445	Internal memory DMA access modifier	Undefined
CSP2B	0x446	Contains number of DMA transfers remaining	Undefined
CPSP2B	0x447	Points to next DMA parameters	Undefined

Register Mnemonic	Address	Description	Reset
IISP3A	0x448	Internal memory DMA address	Undefined
IMSP3A	0x449	Internal memory DMA access modifier	Undefined
CSP3A	0x44A	Contains number of DMA transfers remaining	Undefined
CPSP3A	0x44B	Points to next DMA parameters	Undefined
IISP3B	0x44C	Internal memory DMA address	Undefined
IMSP3B	0x44D	Internal memory DMA access modifier	Undefined
CSP3B	0x44E	Contains number of DMA transfers remaining	Undefined
CPSP3B	0x44F	Points to next DMA parameters	Undefined
TXSP2A	0x460	SPORT 2A transmit data	Undefined
RXSP2A	0x461	SPORT 2A receive data	Undefined
TXSP2B	0x462	SPORT 2B transmit data	Undefined
RXSP2B	0x463	SPORT 2B receive data	Undefined
TXSP3A	0x464	SPORT 3A transmit data	Undefined
RXSP3A	0x465	SPORT 3A receive data	Undefined
TXSP3B	0x466	SPORT 3B transmit data	Undefined
RXSP3B	0x467	SPORT 3B receive data	Undefined
Serial Port 4 and 5 Registers			
SPCTL4	0x800	SPORT 4 control	0x0000 0000
SPCTL5	0x801	SPORT 5 control	0x0000 0000
DIV4	0x802	SPORT 4 divisor for TX/RX SCLK4 and FS4	0x0
DIV5	0x803	SPORT 5 divisor for TX/RX SCLK5 and FS5	0x0
SPMCTL45	0x804	SPORTs 4 & 5 Multichannel Control	0x0
MT4CS0	0x805	SPORT 4 multichannel TX select, channels 31–0	Undefined
MT4CS1	0x806	SPORT 4 multichannel TX select, channels 63–32	Undefined
MT4CS2	0x807	SPORT 4 multichannel TX select, channels 95–64	Undefined
MT4CS3	0x808	SPORT 4 multichannel TX select, channels 127–96	Undefined
MR5CS0	0x809	SPORT 5 multichannel RX select, channels 31–0	Undefined

Register Mnemonic	Address	Description	Reset
MR5CS1	0x80A	SPORT 5 multichannel RX select, channels 63–32	Undefined
MR5CS2	0x80B	SPORT 5 multichannel RX select, channels 95–64	Undefined
MR5CS3	0x80C	SPORT 5 multichannel RX select, channels 127–96	Undefined
MT4CCS0	0x80D	SPORT 4 multichannel TX compand select, channels 31–0	Undefined
MT4CCS1	0x80E	SPORT 4 multichannel TX compand select, channels 63–32	Undefined
MT4CCS2	0x80F	SPORT 4 multichannel TX compand select, channels 95–64	Undefined
MT4CCS3	0x810	SPORT 4 multichannel TX compand select, channels 127–96	Undefined
MR5CCS0	0x811	SPORT 5 multichannel RX compand select, channels 31–0	Undefined
MR5CCS1	0x812	SPORT 5 multichannel RX compand select, channels 63–32	Undefined
MR5CCS2	0x813	SPORT 5 multichannel RX compand select, channels 95–64	Undefined
MR5CCS3	0x814	SPORT 5 multichannel RX compand select, channels 127–96	Undefined
IISP4A	0x840	Internal memory DMA address	Undefined
IMSP4A	0x841	Internal memory DMA access modifier	Undefined
CSP4A	0x842	Contains number of DMA transfers remaining	Undefined
CPSP4A	0x843	Points to next DMA parameters	Undefined
IISP4B	0x844	Internal memory DMA address	Undefined
IMSP4B	0x845	Internal memory DMA access modifier	Undefined
CSP4B	0x846	Contains number of DMA transfers remaining	Undefined
CPSP4B	0x847	Points to next DMA parameters	Undefined
IISP5A	0x848	Internal memory DMA address	Undefined
IMSP5A	0x849	Internal memory DMA access modifier	Undefined
CSP5A	0x84A	Contains number of DMA transfers remaining	Undefined
CPSP5A	0x84B	Points to next DMA parameters	Undefined
IISP5B	0x84C	Internal memory DMA address	Undefined
IMSP5B	0x84D	Internal memory DMA access modifier	Undefined
CSP5B	0x84E	Contains number of DMA transfers remaining	Undefined
CPSP5B	0x84F	Points to next DMA parameters	Undefined

Register Mnemonic	Address	Description	Reset
TXSP4A	0x860	SPORT 4A transmit data	Undefined
RXSP4A	0x861	SPORT 4A receive data	Undefined
TXSP4B	0x862	SPORT 4B transmit data	Undefined
RXSP4B	0x863	SPORT 4B receive data	Undefined
TXSP5A	0x864	SPORT 5A transmit data	Undefined
RXSP5A	0x865	SPORT 5A receive data	Undefined
TXSP5B	0x866	SPORT 5B transmit data	Undefined
RXSP5B	0x867	SPORT 5B receive data	Undefined
SPI Registers			
SPICTL	0x1000	SPI Control	0x0400
SPIFLG	0x1001	SPI Flag	0x0F80
SPISTAT	0x1002	SPI Status	0x01
TXSPI	0x1003	SPI transmit data	Undefined
RXSPI	0x1004	SPI receive data	Undefined
SPIBAUD	0x1005	SPI baud setup	Undefined
RXSPI_SHADOW	0x1006	SPI receive data shadow	Undefined
IISPI	0x1080	Internal memory DMA address	Undefined
IMSPI	0x1081	Internal memory DMA access modifier	Undefined
CSPI	0x1082	Contains number of DMA transfers remaining	Undefined
CPSPI	0x1083	Points to next DMA parameters	Undefined
SPIDMAC	0x1084	SPI DMA control	Undefined
SPIB Registers: This SPI port is routed through the DAI			
SPICTLB	0x2800	SPIB Control	0x0400
SPIFLGB	0x2801	SPIB Flag	0x0F00
SPISTATB	0x2802	SPIB Status	0x01
TXSPIB	0x2803	SPIB transmit data	Undefined
RXSPIB	0x2804	SPIB receive data	Undefined

Register Mnemonic	Address	Description	Reset
SPIBAUDB	0x2805	SPIB baud setup	Undefined
RXSPIB_SHADOW	0x2806	SPIB receive data shadow	Undefined
IISPIB	0x2880	Internal memory DMA address	Undefined
IMSPIB	0x2881	Internal memory DMA access modifier	Undefined
CSPIB	0x2882	Contains number of DMA transfers remaining	Undefined
CPSPIB	0x2883	Points to next DMA parameters	Undefined
SPIDMACB	0x2884	SPIB DMA control	Undefined
Parallel Port Registers			
PPCTL	0x1800	Parallel port control	0x0000 402E
RXPP	0x1808	Parallel port receive data	Undefined
TXPP	0x1809	Parallel port transmit data	Undefined
EIPP	0x1810	External memory DMA address	Undefined
EMPP	0x1811	External memory DMA access modifier	Undefined
ECPP	0x1812	Contains number of external DMA accesses remaining	Undefined
IIPP	0x1818	Internal memory DMA address	Undefined
IMPP	0x1819	Internal memory DMA access modifier	Undefined
ICPP	0x181A	Contains number of DMA transfers remaining	Undefined
CPPP	0x181B	Parallel port chain pointer	Undefined
Timer Registers			
TMSTAT	0x1400	GP Timer Status Register. TMxSTAT all address the same register TMSTAT	0x0
TM0STAT	0x1400	GP Timer 0 Status	0x0
TM0CTL	0x1401	GP Timer 0 Control	0x0
TM0CNT	0x1402	GP Timer 0 Count	0x0
TM0PRD	0x1403	GP Timer 0 Period	0x0
TM0W	0x1404	GP Timer 0 Width	0x0
TM1STAT	0x1408	GP Timer 1 Status	0x0

Register Mnemonic	Address	Description	Reset
TM1CTL	0x1409	GP Timer 1 Control	0x0
TM1CNT	0x140A	GP Timer 1 Count	0x0
TM1PRD	0x140B	GP Timer 1 Period	0x0
TM1W	0x140C	GP Timer 1 Width	0x0
TM2STAT	0x1410	GP Timer 2 Status	0x0
TM2CTL	0x1411	GP Timer 2 Control	0x0
TM2CNT	0x1412	GP Timer 2 Count	0x0
TM2PRD	0x1413	GP Timer 2 Period	0x0
TM2W	0x1414	GP Timer 2 Width	0x0
Power Management Registers			
PMCTL	0x2000	Power management control	HW related
DMA Parameter Registers			
IDP_DMA_I0	0x2400	IDP DMA Channel 0 Index	0x0
IDP_DMA_I1	0x2401	IDP DMA Channel 1 Index	0x0
IDP_DMA_I2	0x2402	IDP DMA Channel 2 Index	0x0
IDP_DMA_I3	0x2403	IDP DMA Channel 3 Index	0x0
IDP_DMA_I4	0x2404	IDP DMA Channel 4 Index	0x0
IDP_DMA_I5	0x2405	IDP DMA Channel 5 Index	0x0
IDP_DMA_I6	0x2406	IDP DMA Channel 6 Index	0x0
IDP_DMA_I7	0x2407	IDP DMA Channel 7 Index	0x0
IDP_DMA_I0A	0x2408	IDP DMA Channel 0 Index A for Ping Pong DMA	0x0
IDP_DMA_I1A	0x2409	IDP DMA Channel 1 Index A for Ping Pong DMA	0x0
IDP_DMA_I2A	0x240A	IDP DMA Channel 2 Index A for Ping Pong DMA	0x0
IDP_DMA_I3A	0x240B	IDP DMA Channel 3 Index A for Ping Pong DMA	0x0
IDP_DMA_I4A	0x240C	IDP DMA Channel 4 Index A for Ping Pong DMA	0x0
IDP_DMA_I5A	0x240D	IDP DMA Channel 5 Index A for Ping Pong DMA	0x0
IDP_DMA_I6A	0x240E	IDP DMA Channel 6 Index A for Ping Pong DMA	0x0

Register Mnemonic	Address	Description	Reset
IDP_DMA_I7A	0x240F	IDP DMA Channel 7 Index A for Ping Pong DMA	0x0
IDP_DMA_I0B	0x2418	IDP DMA Channel 0 Index B for Ping Pong DMA	0x0
IDP_DMA_I1B	0x2419	IDP DMA Channel 1 Index B for Ping Pong DMA	0x0
IDP_DMA_I2B	0x241A	IDP DMA Channel 2 Index B for Ping Pong DMA	0x0
IDP_DMA_I3B	0x241B	IDP DMA Channel 3 Index B for Ping Pong DMA	0x0
IDP_DMA_I4B	0x241C	IDP DMA Channel 4 Index B for Ping Pong DMA	0x0
IDP_DMA_I5B	0x241D	IDP DMA Channel 5 Index B for Ping Pong DMA	0x0
IDP_DMA_I6B	0x241E	IDP DMA Channel 6 Index B for Ping Pong DMA	0x0
IDP_DMA_I7B	0x241F	IDP DMA Channel 7 Index B for Ping Pong DMA	0x0
IDP_DMA_M0	0x2410	IDP DMA Channel 0 Modify	0x0
IDP_DMA_M1	0x2411	IDP DMA Channel 1 Modify	0x0
IDP_DMA_M2	0x2412	IDP DMA Channel 2 Modify	0x0
IDP_DMA_M3	0x2413	IDP DMA Channel 3 Modify	0x0
IDP_DMA_M4	0x2414	IDP DMA Channel 4 Modify	0x0
IDP_DMA_M5	0x2415	IDP DMA Channel 5 Modify	0x0
IDP_DMA_M6	0x2416	IDP DMA Channel 6 Modify	0x0
IDP_DMA_M7	0x2417	IDP DMA Channel 7 Modify	0x0
IDP_DMA_C0	0x2420	IDP DMA Channel 0 Count	0x0
IDP_DMA_C1	0x2421	IDP DMA Channel 1 Count	0x0
IDP_DMA_C2	0x2422	IDP DMA Channel 2 Count	0x0
IDP_DMA_C3	0x2423	IDP DMA Channel 3 Count	0x0
IDP_DMA_C4	0x2424	IDP DMA Channel 4 Count	0x0
IDP_DMA_C5	0x2425	IDP DMA Channel 5 Count	0x0
IDP_DMA_C6	0x2426	IDP DMA Channel 6 Count	0x0
IDP_DMA_C7	0x2427	IDP DMA Channel 7 Count	0x0
IDP_DMA_PC0	0x2428	IDP DMA Channel 0 Ping Pong Count	0x0
IDP_DMA_PC1	0x2429	IDP DMA Channel 1 Ping Pong Count	0x0

Register Mnemonic	Address	Description	Reset
IDP_DMA_PC2	0x242A	IDP DMA Channel 2 Ping Pong Count	0x0
IDP_DMA_PC3	0x242B	IDP DMA Channel 3 Ping Pong Count	0x0
IDP_DMA_PC4	0x242C	IDP DMA Channel 4 Ping Pong Count	0x0
IDP_DMA_PC5	0x242D	IDP DMA Channel 5 Ping Pong Count	0x0
IDP_DMA_PC6	0x242E	IDP DMA Channel 6 Ping Pong Count	0x0
IDP_DMA_PC7	0x242F	IDP DMA Channel 7 Ping Pong Count	0x0
SRU Registers			
SRU_CLK0	0x2430	SRU Clock Control 0	0x2526 30C2
SRU_CLK1	0x2431	SRU Clock Control 1	0x3DEF 7BDE
SRU_CLK2	0x2432	SRU Clock Control 2	0x3DEF 7BDE
SRU_CLK3	0x2433	SRU Clock Control 3	0x3DEF 7BDE
SRU_CLK4	0x2434	SRU Clock Control 4	0x3DEF 7BDE
SRU_DAT0	0x2440	SRU Data Control 0	0x0814 4040
SRU_DAT1	0x2441	SRU Data Control 1	0x0F38 B289
SRU_DAT2	0x2442	SRU Data Control 2	0x0000 0450
SRU_DAT3	0x2443	SRU Data Control 3	0x0
SRU_DAT4	0x2444	SRU Data Control 4	0x0
SRU_DAT5	0x2445	SRU Data Control 5	0x0
SRU_FS0	0x2450	SRU FS Control 0	0x2736 B4E3
SRU_FS1	0x2451	SRU FS Control 1	0x3DEF 7BDE
SRU_FS2	0x2452	SRU FS Control 2	0x3DEF 7BDE
SRU_FS3	0x2453	SRU FS Control 3	0x3DEF 7BDE
SRU_PIN0	0x2460	SRU Pin Control 0	0x04C8 0A94
SRU_PIN1	0x2461	SRU Pin Control 1	0x04E8 4B96
SRU_PIN2	0x2462	SRU Pin Control 2	0x0366 8C98
SRU_PIN3	0x2463	SRU Pin Control 3	0x03A7 14A3
SRU_PIN4	0x2464	SRU Pin Control 4	0x0569 4F9E

Register Mnemonic	Address	Description	Reset
SRU_EXT_MISCA	0x2470	SRU External Misc. A Control	0x3DEF 7BDE
SRU_EXT_MISCB	0x2471	SRU External Misc. B Control	0x3DEF 7BDE
SRU_PBEN0	0x2478	SRU Pin Enable 0	0x0E24 82CA
SRU_PBEN1	0x2479	SRU Pin Enable 1	0x1348 D30F
SRU_PBEN2	0x247A	SRU Pin Enable 2	0x1A55 45D6
SRU_PBEN3	0x247B	SRU Pin Enable 3	0x1D71 F79B
DAI_PIN_PULLUP	0x247D	Controls whether DAI bin buffers have pullups enabled	0xFFFFF
DAI Interrupt Registers			
DAI_IRPTL_FE	0x2480	DAI Falling Edge Interrupt Latch	0x0
DAI_IRPTL_RE	0x2481	DAI Rising Edge Interrupt Latch	0x0
DAI_IRPTL_PRI	0x2484	DAI Interrupt Priority	0x0
DAI_IRPTL_H	0x2488	DAI High Priority Interrupt Latch	0x0
DAI_IRPTL_L	0x2489	DAI Low Priority Interrupt Latch	0x0
DAI_IRPTL_HS	0x248C	Shadow DAI High Priority Interrupt Latch	0x0
DAI_IRPTL_LS	0x248D	Shadow DAI Low Priority Interrupt Latch	0x0
Input Data Port Registers			
IDP_CTL0	0x24B0	IDP Control 0	0x0
IDP_PP_CTL	0x24B1	IDP Parallel Port Control	0x0
IDP_CTL1	0x24B2	IDP Control 1	0x0000 FFFF
IDP_FIFO	0x24D0	IDP FIFO Packing Mode	0x0
DAI Status Registers			
DAI_STAT	0x24B8	DAI Status	0x0
DAI_PIN_STAT	0x24B9	DAI Pin Buffer Status	0x000F FFFF
Precision Clock Generator Registers			
PCG_CTLA0	0x24C0	Precision Clock A Control 0	0x0
PCG_CTLA1	0x24C1	Precision Clock A Control 1	0x0
PCG_CTLB0	0x24C2	Precision Clock B Control 0	0x0

Register Mnemonic	Address	Description	Reset
PCG_CTLB1	0x24C3	Precision Clock B Control 1	0x0
PCG_PW	0x24C4	Precision Clock Pulse Width Control	0x0
PCG_SYNC	0x24C5	Precision Clock Frame Sync Synchronization	0x0
Peripheral Interrupt Priority Control Registers			
PICR0	0x2200	Peripheral Interrupt Priority Control 0	0x0A41 8820
PICR1	0x2201	Peripheral Interrupt Priority Control 1	0x16A4 A0E6
PICR2	0x2202	Peripheral Interrupt Priority Control 2	0x2307 B9AC
PICR3	0x2203	Peripheral Interrupt Priority Control 3	0x0000 0012
Pulse Width Modulation Registers			
PWMGCTL	0x3800	PWM Global Control	0x0
PWMGSTAT	0x3801	PWM Global Status	0x0
PWMCTL0	0x3000	PWM Control 0	0x0
PWMSTAT0	0x3001	PWM Status 0	0x0009
PWMPERIOD0	0x3002	PWM Period 0	0x0
PWMDT0	0x3003	PWM Dead Time 0	0x0
PWMA0	0x3005	PWM Channel A Duty Control 0	0x0
PWMB0	0x3006	PWM Channel B Duty Control 0	0x0
PWMSEG0	0x3008	PWM Output Enable 0	0x0
PWMAL0	0x300A	PWM Channel AL Duty Control 0	0x0
PWMBL0	0x300B	PWM Channel BL Duty Control 0	0x0
PWMDBG0	0x300E	PWM Debug Status 0	0x0
PWMPOL0	0x300F	PWM Output Polarity Select 0	0x00FF
PWMCTL1	0x3010	PWM Control 1	0x0
PWMSTAT1	0x3011	PWM Status 1	0x0009
PWMPERIOD1	0x3012	PWM Period 1	0x0
PWMDT1	0x3013	PWM Dead Time 1	0x0
PWMA1	0x3015	PWM Channel A Duty Control 1	0x0

Register Mnemonic	Address	Description	Reset
PWMB1	0x3016	PWM Channel B Duty Control 1	0x0
PWMSEG1	0x3018	PWM Output Enable 1	0x0
PWMAL1	0x301A	PWM Channel AL Duty Control 1	0x0
PWMBL1	0x301B	PWM Channel BL Duty Control 1	0x0
PWMDBG1	0x301E	PWM Debug Status 1	0x0
PWMPOL1	0x301F	PWM Output Polarity Select 1	0x00FF
PWMCTL2	0x3400	PWM Control 2	0x0
PWMSTAT2	0x3401	PWM Status 2	0x0009
PWMPERIOD2	0x3402	PWM Period 2	0x0
PWMDT2	0x3403	PWM Dead Time 2	0x0
PWMA2	0x3405	PWM Channel A Duty Control 2	0x0
PWMB2	0x3406	PWM Channel B Duty Control 2	0x0
PWMSEG2	0x3408	PWM Output Enable 2	0x0
PWMAL2	0x340A	PWM Channel AL Duty Control 2	0x0
PWMBL2	0x340B	PWM Channel BL Duty Control 2	0x0
PWMDBG2	0x340E	PWM Debug Status 2	0x0
PWMPOL2	0x340F	PWM Output Polarity Select 2	0x00FF
PWMCTL3	0x3410	PWM Control 3	0x0
PWMSTAT3	0x3411	PWM Status 3	0x0009
PWMPERIOD3	0x3412	PWM Period 3	0x0
PWMDT3	0x3413	PWM Dead Time 3	0x0
PWMA3	0x3415	PWM Channel A Duty Control 3	0x0
PWMB3	0x3416	PWM Channel B Duty Control 3	0x0
PWMSEG3	0x3418	PWM Output Enable 3	0x0
PWMAL3	0x341A	PWM Channel AL Duty Control 3	0x0
PWMBL3	0x341B	PWM Channel BL Duty Control 3	0x0
PWMDBG3	0x341E	PWM Debug Status 3	0x0

Register Mnemonic	Address	Description	Reset
PWMPOL3	0x341F	PWM Output Polarity Select 3	0x00FF
Memory-to-Memory DMA Registers			
MTMCTL	0x2C01	Memory-to-Memory DMA Control	0x0
IIMTMW	0x2C10	MTM DMA Destination Index	0x0
IIMTMR	0x2C11	MTM DMA Source Index	0x0
IMMTMW	0x2C0E	MTM DMA Destination Modify	0x0
IMMTMR	0x2C0F	MTM DMA Source Modify	0x0
CMTMW	0x2C1	MTM DMA Destination Count	0x0
CMTMR	0x2C17	MTM DMA Source Count	0x0
Sample Rate Converter Registers			
SRCCTL0	0x2490	SRC0 Control	0x0
SRCCTL1	0x2491	SRC1 Control	0x0
SRCMUTE	0x2492	SRC Mute	0x0
SRCRAT0	0x2498	SRC0 Output to Input Ratio	0x8000 8000
SRCRAT1	0x2499	SRC1 Output to Input Ratio	0x8000 8000

B INTERRUPTS

This chapter provides a listing of the registers that are used to configure and control programmable interrupts. For information on interrupt vector tables and the core interrupt registers, see the *SHARC Processor Programming Reference*.

Programmable Interrupt Control Registers

The following sections provide descriptions of the programmable interrupts that are used in the ADSP-2136x processors. These registers allow programs to substitute the default interrupts for some other interrupt source. For information on the interrupt registers and the interrupt vector table, see Appendix B, Interrupts.

Table B-1 lists the locations to be programmed in the programmable interrupt control registers (`PICR`) to route a peripheral interrupt source to a corresponding processor interrupt location.

Table B-1 also defines the `PICR` bits which should be programmed to select the source for each priority interrupt. Priority programming can be accomplished by changing the sources for each priority interrupt. For

example, if peripheral x should be given high priority, the high priority priority interrupt source should be set as that peripheral (x).

Table B-1. Default Programmable Interrupt Controller Routing Table

Interrupt Name	Vector Address	Programmable Interrupt Control Register (PICR)	Default Select Value	Default Function	Priority
P0I	0x2C	PICR0[4–0]	0x00	DAIHI interrupt	HIGHEST
P1I[1]	0x30	PICR0[9–5]	0x01	SPI1 high interrupt	
P2I	0x34	PICR0[14–10]	0x02	GP timer-0 interrupt	
P3I	0x38	PICR0[19–15]	0x03	SPORT1 interrupt	
P4I	0x3C	PICR0[24–20]	0x04	SPORT3 interrupt	
P5I	0x40	PICR0[29–25]	0x05	SPORT5 interrupt	
P6I	0x44	PICR1[4–0]	0x06	SPORT0 interrupt	
P7I	0X48	PICR1[9–5]	0x07	SPORT2 interrupt	
P8I	0X4C	PICR1[14–10]	0x08	SPORT4 interrupt	
P9I[1]	0X50	PICR1[19–15]	0x09	Parallel port interrupt	
P10I	0X54	PICR1[24–20]	0x0A	GP Timer-1 interrupt	
P11I	0x58	PICR1[29–25]	0x0B	Reserved	
P12I	0x5C	PICR2[4–0]	0x0C	DAILI interrupt	
P13I	0x60	PICR2[9–5]	0x0D	PWM interrupt	
P14I	0x64	PICR2[14–10]	0x0E	Reserved	
P15I	0x68	PICR2[19–15]	0x0F	MTM, DTCP interrupt	
P16I	0x6C	PICR2[24–20]	0x10	Reserved	
P17I	0x70	PICR2[29–25]	0x11	GP timer-2 interrupt	
P18I	0x74	PICR3[4–0]	0x12	SPIB low interrupt	LOWEST

1 These interrupts have an option to be unmasked at reset. Therefore, the peripherals that boot the processor should be allocated these interrupts: (P1I, P9I).

Programmable Interrupt Control Register 0 (PICR0)

This 32-bit read/write register, shown in Figure B-1, controls programmable priority interrupts 0–5 and the default sources.

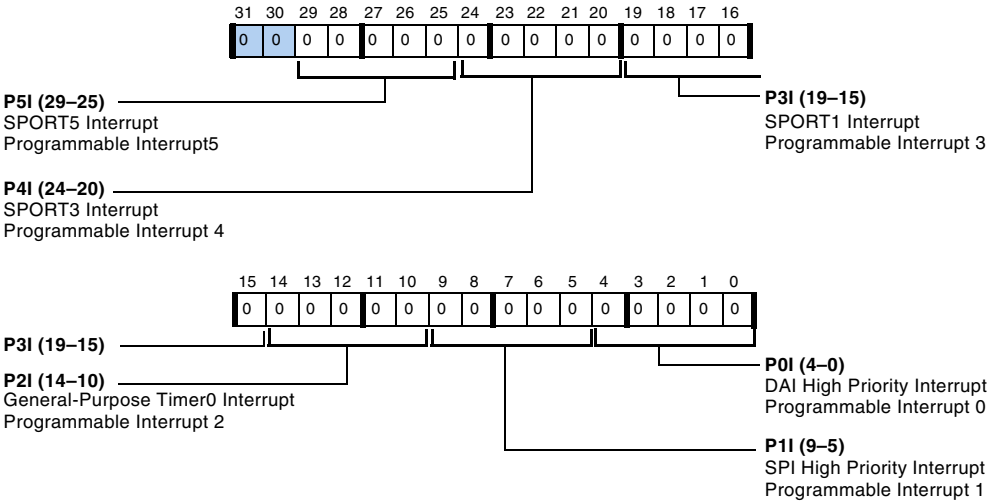

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
P5I (29–25)
SPORT5 Interrupt
Programmable Interrupt5
P4I (24–20)
SPORT3 Interrupt
Programmable Interrupt 4
P3I (19–15)
SPORT1 Interrupt
Programmable Interrupt 3
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
P3I (19–15)
P2I (14–10)
General-Purpose Timer0 Interrupt
Programmable Interrupt 2
P0I (4–0)
DAI High Priority Interrupt
Programmable Interrupt 0
P1I (9–5)
SPI High Priority Interrupt
Programmable Interrupt 1

Figure B-1. PICR0 Register

Programmable Interrupt Control Register 1 (PICR1)

This register, shown in Figure B-2, controls programmable peripheral interrupts 6–11 and the default sources.

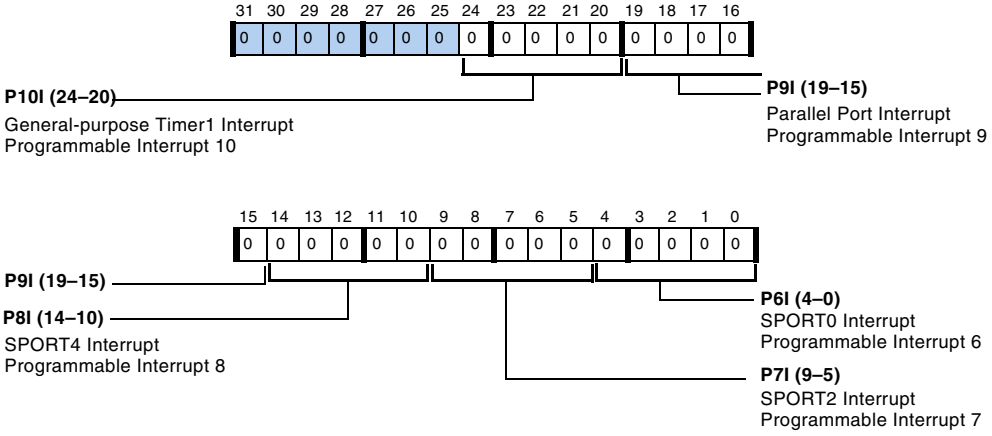

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
P10I (24–20)
General-purpose Timer1 Interrupt
Programmable Interrupt 10
P9I (19–15)
Parallel Port Interrupt
Programmable Interrupt 9
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
P9I (19–15)
P8I (14–10)
SPORT4 Interrupt
Programmable Interrupt 8
P6I (4–0)
SPORT0 Interrupt
Programmable Interrupt 6
P7I (9–5)
SPORT2 Interrupt
Programmable Interrupt 7

Figure B-2. PICR1 Register

Programmable Interrupt Control Register 2 (PICR2)

This register, shown in Figure B-3, controls programmable peripheral interrupts 12 through 17 as well as the default sources.

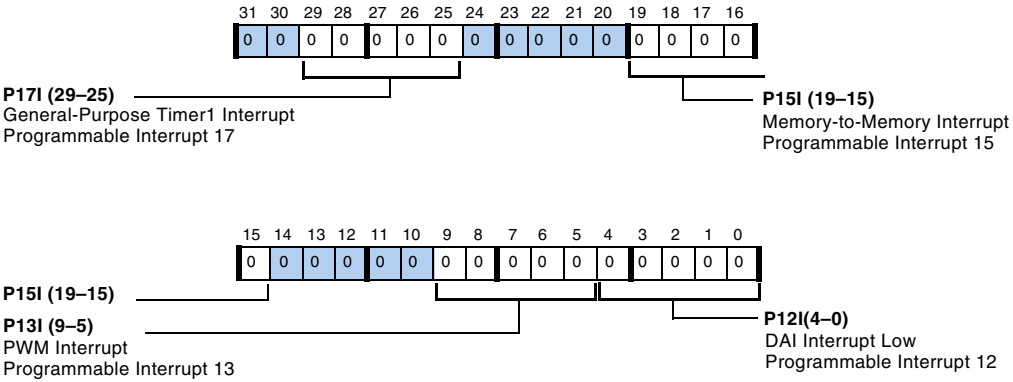

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
P17I (29–25)
General-Purpose Timer1 Interrupt
Programmable Interrupt 17
P15I (19–15)
Memory-to-Memory Interrupt
Programmable Interrupt 15
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
P15I (19–15)
P13I (9–5)
PWM Interrupt
Programmable Interrupt 13
P12I(4–0)
DAI Interrupt Low
Programmable Interrupt 12

Figure B-3. PICR2 Register

Programmable Interrupt Control Register 3 (PICR3)

This register, shown in Figure B-4, controls programmable peripheral interrupt 18.

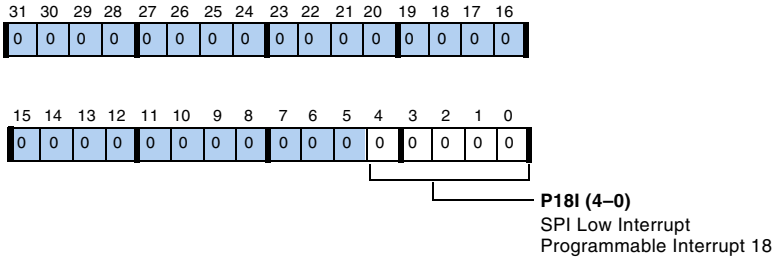


Figure B-4. PICR3 Register

C AUDIO FRAME FORMATS

This appendix introduces all the serial timing protocols used for audio inter-chip communications. These formats are listed and their availability in the various peripherals noted in Table C-1.

Table C-1. Audio Format Availability

Frame Format	SPORTs	IDP/SIP	ASRC Input	ASRC Output	S/PDIF Tx	S/PDIF Rx	PCG
Serial	Yes						Yes
I^2S	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Left-justified	Yes	Yes	Yes	Yes	Yes		Yes
Right-justified, 24-bit		Yes	Yes	Yes	Yes		Yes
Right-justified, 20-bit		Yes	Yes	Yes	Yes		Yes
Right-justified, 18-bit		Yes	Yes	Yes	Yes		Yes
Right-justified, 16-bit		Yes	Yes	Yes	Yes		Yes
TDM, 128 channel	Yes		Yes	Yes			Yes

Overview

The following protocols are available in the SHARC processor and are briefly described in this appendix. For complete information on the industry standard protocols, see the specification listings in each section.

- Standard Serial Mode
- Left-justified Mode (Sony format)
- I^2S Mode (Sony/Philips format)
- Time Division Multiplex (TDM) Mode
- MOST Mode
- Right-justified Mode
- S/PDIF (consumer mode)
- EBU/AES3 (professional mode)

Standard Serial Mode

Most processors allow word lengths of 4 to 32 bits to be transmitted or received through their serial ports. For convenience, most AFE (analog front-end) devices operate with 16-bit word lengths for both data and status transfer between the AFE and processor. The serial ports (SPORTs) of most DSPs are designed for full-duplex operation. They differ from the typical serial interface of micro controllers in that they use a frame sync pulse to indicate the start of the data frame. In the case of full duplex asynchronous transfers, two separate FS pulses are used for transmit and receive. The typical micro controller serial interfaces use the serial clock (SCLK) as an indicator of serial data, meaning that the SCLK is only active when data is valid. The DSP serial interface can operate with a continuous

SCLK, in which the frame synchronization (FS) pulse indicates the start of valid data.

Serial mode allows a flexible timing which can be used in unframed mode or framed mode. In framed mode the user can select between timing for early and late frame sync. Moreover the word order can be selected as LSB or MSB first.

I²S Mode

The Inter-IC-Sound (I^2S) bus protocol is a popular 3 wire serial bus standard that was developed to standardize communication across a wide range of peripheral devices. Today the I^2S protocol has become the standard method of communicating with consumer and professional audio products.

The I^2S protocol provides transmission of 2 channel (stereo) Pulse Code Modulation digital data, where each audio sample is sent MSB first. The following list shows applications that use this format.

- Audio D/A and A/D converters
- PC multimedia audio controllers
- Digital audio transmitters and receivers that support serial digital audio transmission standards, such as AES/EBU, S/PDIF, IEC958, CP-340, and CP-1201
- Digital audio signal processors
- Dedicated digital filter chips
- Sample rate converters

Timing diagrams for I^2S, right-justified and left-justified formats can be found in the product specific data sheet.

The I²S bus transmits audio data from 8–32 bits and control signals over separate lines. The data line carries two multiplexed data channels—the left channel and the right channel. In I²S mode, if both channels on a SPORT are set up to transmit, then the SPORT transmits left and right I²S channels simultaneously. If both channels on a SPORT are set up to receive, the SPORT receives left and right I²S channels simultaneously. Data is transmitted in MSB-first format.

I²S consists, as stated above, of a bit clock, a word select and the data line. The bit clock pulses once for each discrete bit of data on the data lines. The bit clock operates at a frequency which is a multiple of the sample rate. The bit clock frequency multiplier depends on number of bits per channel, times the number of channels. For example, CD Audio with a sample frequency of 44.1 kHz and 32 bits of precision per (2) stereo channels has a bit clock frequency of 2.8224 MHz. The word select clock lets the device know whether channel 1 or channel 2 is currently being sent, since I²S allows two channels to be sent on the same data line.

Transitions on the word select clock also serve as a start-of-word indicator. The word clock line pulses once per sample, so while the bit clock runs at some multiple of the sample frequency, the word clock always matches the sample frequency. For a two channel (stereo) system, the word clock is a square wave, with an equal number of bit clock pulses clocking the data to each channel. In a mono system, the word clock pulses one bit clock length to signal the start of the next word, but is no longer be square. Instead, bit clocking transitions occur with the word clock either high or low.

Note the major difference between I²S and left/right justified modes is a left MSB data shift by one `SCLK` cycle in relation to the frame.

Standard I²S data is sent from MSB to LSB, starting at the left edge of the word select clock, with one bit clock delay. This allows both the transmitting and receiving devices to ignore the audio precision of the remote

device. If the transmitter is sending 32 bits per channel to a device with only 24 bits of internal precision, the receiver ignores the extra bits of precision by not storing the bits past the 24th bit. Likewise, if the transmitter is sending 16 bits per channel to a receiving device with 24 bits of precision, the receiver zero-fills the missing bits. This feature makes it possible to mix and match components of varying precision without re configuration.

Left-Justified Mode

Left-justified mode (also known as SONY Format) is a mode where in each frame sync cycle two samples of data are transmitted/received—one sample on the high segment of the frame sync, the other on the low segment of the frame sync. Prior to development of the I^2S standard, many manufacturers used a variety of non-standard stereo modes. Some companies continue to use this mode, which is supported by many of today's audio front-end devices.

Programs have control over various attributes of this mode. One attribute is the number of bits (8-to 32-bit word lengths). However, each sample of the pair that occurs on each frame sync must be the same length.

Right-Justified Mode

Right-justified mode is a mode where in each frame sync cycle two samples of data are transmitted/received—one sample on the high segment of the frame sync, the other on the low segment of the frame sync. Prior to development of the I^2S standard, many manufacturers used a variety of non-standard stereo modes. Some companies continue to use this mode, which is supported by many of today's audio front-end devices.

Programs have control over various attributes of this mode. One attribute is the number of bits (8-to 32-bit word lengths). However, each sample of the pair that occurs on each frame sync must be the same length.

TDM Mode

Many applications require multiple I/O channels to implement the desired system functions (such as telephone line and acoustic interfaces). Because most DSPs provide one, or at most two SPORTs, and one of these may be required for interfacing to the host or supervisory processor, it may be impractical, if not impossible, to dedicate a separate SPORT interface to each AFE connection.

The solution is to devise a way to connect a series of serial devices to one SPORT. Different converter manufacturers have approached this task in different ways. In essence, though, there are only two choices; either a time division multiplexing (TDM) approach, where each device is active on the SPORT in a particular time slot, or a cascading approach, where all devices are daisy chained together and data is transferred by shifting it through the chain and then following with a latching signal or a serial protocol. Figure C-1 illustrates a pulsed frame clock for the TDM operation.

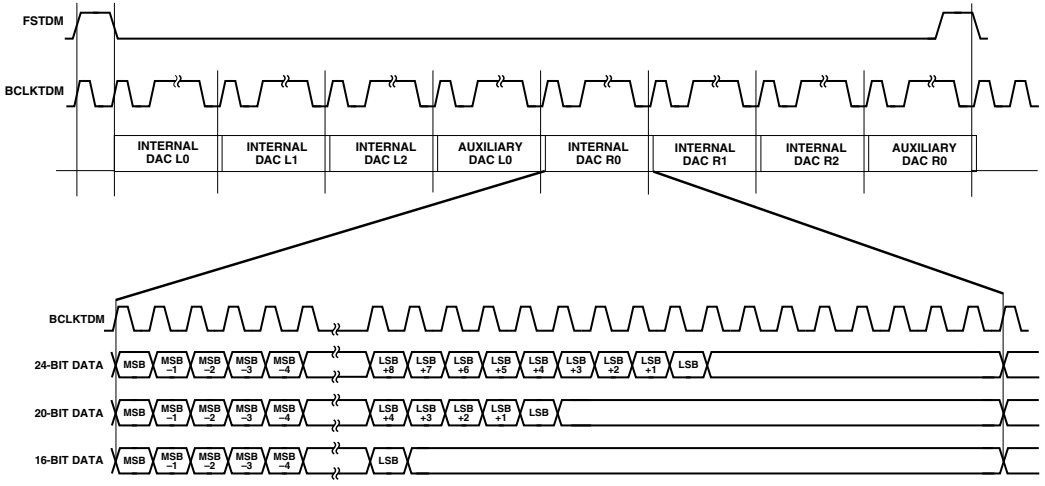


Figure C-1. TDM Mode Timing

Packed TDM Mode

This mode allows applications to send more than the standard 32 bits per channel available through standard I^2S mode. Packed mode is implemented using standard TDM mode. Packed mode supports up to 128 channels as does TDM mode as well as the maximum of (128 x 32) bits per left or right channel. As shown in Figure C-2 packed I^2S waveforms are the same as the wave forms used in TDM mode, except that the frame sync is toggled for every frame, and therefore emulates I^2S mode. So it is a hybrid between TDM and I^2S mode.

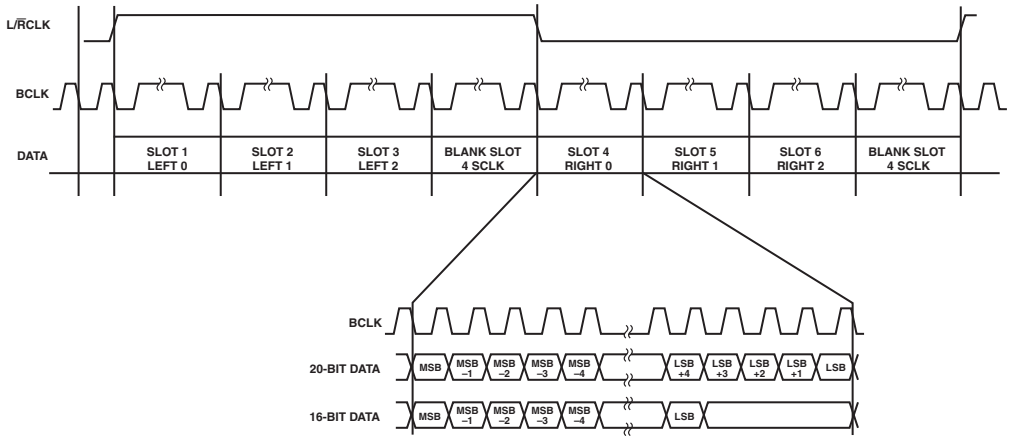


Figure C-2. Packed TDM Mode

MOST Mode

A special packed TDM mode is available that allows four channels to be fit into a space of 64-bit clock cycles. This mode is called packed TDM4 mode, or MOST™ mode. MOST (MediaOriented Systems Transport) is a networking standard intended for interconnecting multimedia components in automobiles and other vehicles. Many integrated circuits intended to interface with a MOST bus use a packed TDM4 data format.

Figure C-3 illustrates a word length of 16 bits for a timing diagram of the packed TDM4 mode. This figure is shown with a negative`BCLK` polarity, a negative `LRCLK` polarity, and an MSB delay of 1. The MSB position of the serial data must be delayed by one bit clock from the start of the frame (I^2S position).

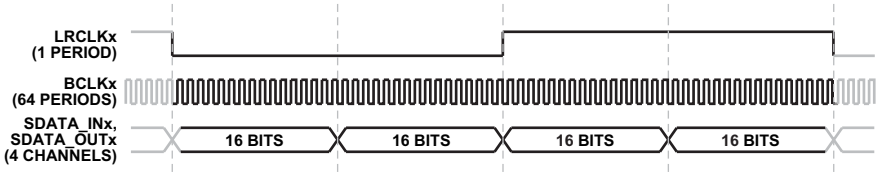


Figure C-3. Packed TDM4 Mode

AES/EBU/SPDIF Formats

For this section, it is important to be familiar with serial digital application interface standards IEC-60958, EIAJ CP-340, AES3 and AES11.

S/PDIF data is transmitted as a stream of 32-bit data words. A data frame consists of 384 words in total, with 192 data words transmitted for the A stereo channel, and 192 data words transmitted for the B stereo channel.

The difference between the AES/EBU and S/PDIF protocol is the channel status bit. If the channel status bit is not set, then:

- 0 = Consumer/professional
- 1 = Normal/compressed data
- 2 = Copy prohibit/copy permit
- 3 = 2 channels/4 channels
- 4 = n/a
- 5 = No pre-emphasis/pre-emphasis

There is one channel status bit in each sub-frame, (comprising of 192 bits per audio block). This translates to 192/8 = 24 bytes available (per audio block). The meaning of the channel status bits are as follows

- The biphase encoded AES3 stream is composed of subframes (Figure C-5 on page C-12). Subframes consist of a preamble, four auxiliary bits, a 20-bit audio word, a validity bit, a user bit, a channel status bit, and a parity bit.
- The preamble indicates the start of the subframe. The four auxiliary bits normally are the least significant bits of the 24-bit audio word when pasted to the 20-bit audio word. In some cases, the auxiliary bits are used to convey some kind of other data indicated by the channel status bits.
- The validity bit (if cleared, =0) indicates the audio sample word is suitable for direct analog conversion. User data bits may be used in any way desired by the program. The channel status bit conveys information about the status of the channel. Examples of status are length of audio sample words, number of audio channels, sampling frequency, sample address code, alphanumeric source, and destination codes and emphasis. The parity bit is set or cleared to provide an even number of ones and of zeros for time slots 4-31.
- Each frame in the AES3 stream is made up of two subframes. The first subframe is channel A, and the second subframe is channel B. A block is comprised of 192 frames. The channel status is organized into two 192 bit blocks, one for channel A and one for channel B. Normally, the channel status of channel A is equal to channel B. It is extremely rare that they are ever different. Three different preambles are used to indicate the start of a block and the start of channel A or B.

1. Preamble Z indicates the start of a block and the start of subframe channel A
2. Preamble X indicates the start of a channel A subframe when not at the start of a block.
3. Preamble Y indicates the start of a channel B subframe.

The user bits from the channel A and B subframes are simply strung together. For more information, please refer to the AES3 standard.

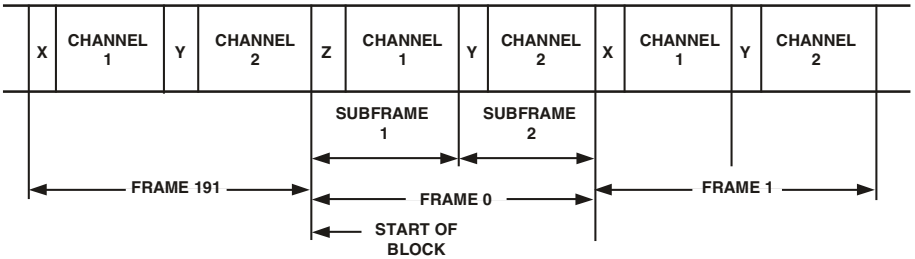


Figure C-4. S/PDIF Block Structure

The data carried by the SPDIF interfaceis transmitted serially. In order to identify the assorted bits of information the data stream is divided into frames, each of which are 64 time slots (or 128 unit intervals[1]) in length (Figure C-4). Since the time slots correspond with the data bits, the frame is often described as being 64 bits in length.

A frame is uniquely composed of two subframes. The first subframe normally starts with preamble X. However, the preamble changes to preamble Z once every 192 frames. This defines the block of frames structure used to organize the channel status information. The second subframe always starts with preamble Y.

Subframe Format

Each frame consists of two subframes. Figure C-5 shows an illustration of a subframe, which consists of 32 time slots numbered 0 to 31. A subframe is 64 unit intervals in length. The first four time slots of each subframe carry the preamble information. The preamble marks the subframe start

[1] The unit interval is the minimum time interval between condition changes of a data transmission signal.

and identifies the subframe type. The next 24 time slots carry the audio sample data, which is transmitted in a 24-bit word with the least significant bit (LSB) first. When a 20-bit coding range is sufficient, time slots 8 to 27 carry the audio sample word with the LSB in time slot 8. Time slots 4 to 7 may be used for other applications. Under these circumstances, the bits in time slots 4 to 7 are designated auxiliary sample bits. If the source provides fewer bits than the interface allows (either 20 or 24), the unused LSBs are set to logic 0.

This functionality is important when using the SPDIF receiver in common applications where there are multiple types of data to handle. If there are PCM audio data streams as well as encoded data streams, for example a CD audio stream and a DVD audio stream with encoded data, there is a danger of incorrectly passing the encoded data directly to the DAC. This results in the 'playing' of encoded data as audio, causing loud odd noises to be played. The non-audio flag provides an easy method to mark the this type of data.

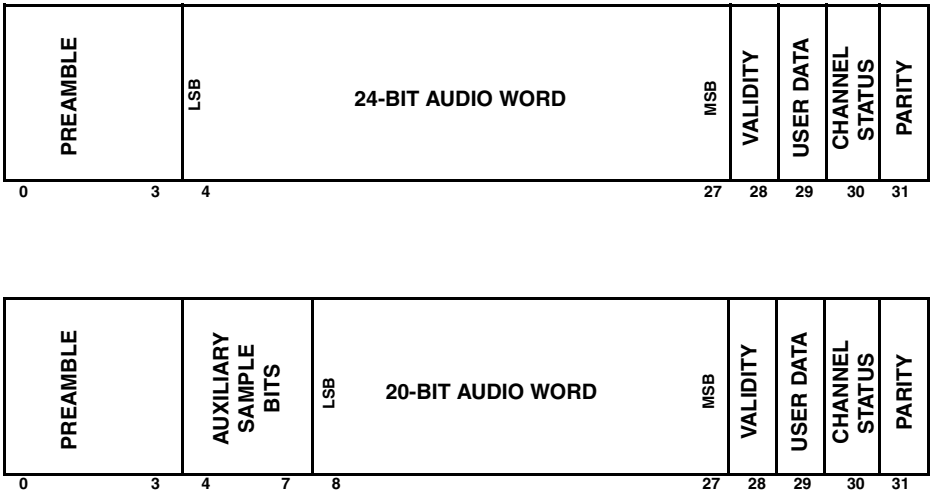


Figure C-5. Subframe Format

After the audio sample word, there are four final time slots which carry:

1. **Validity bit (time slot 28).** The validity bit is logic 0 if the audio sample word is suitable for conversion to an analog audio signal, and logic 1 if it is not. This bit is set if the CHST_BUF_ENABLE bit and the VALIDITY_A (VALIDITY_B for channel 2) bit is set in the SPDIF_TX_CTL register. This bit is also set if the corresponding bit given with the sample is set.

2. **User data bit (time slot 29).** This bit carries user-specified information that may be used in any way. This bit is set if the corresponding bit given with the left/right sample is set.

3. **Channel status bit (time slot 30).** The channel status for each audio signal carries information associated with that audio signal, making it possible for different channel status data to be carried in the two subframes of the digital audio signal. Examples of information to be carried in the channel status are: length of audio sample words, number of audio channels, sampling frequency, sample address code, alphanumeric source and destination codes, and emphasis.

   Channel status information is organized in 192-bit blocks, subdivided into 24 bytes. The first bit of each block is carried in the frame with preamble Z.

4. **Parity bit (time slot 31).** The parity bit indicates that time slots 4 to 31 inclusive will carry an even number of ones and an even number of zeros (even parity). The parity bit is automatically generated for each subframe and inserted into the encoded data.

The two subframes in a frame can be used to transmit two channels of data (channel 1 in subframe 1, channel 2 in subframe 2) with a sample rate equal to the frame rate. Alternatively, the two subframes can carry successive samples of the same channel of data, but at a sample rate that is twice the frame rate. This is called single-channel, double-frequency (SCDF). For more information, see “Output Data Mode” on page 11-8.

Channel Coding

To minimize the direct-current (dc) component on the transmission line, to facilitate clock recovery from the data stream, and to make the interface insensitive to the polarity of connections, time slots 4 to 31 are encoded in bi-phase mark.

Each bit to be transmitted is represented by a symbol comprising two consecutive binary states. The first state of a symbol is always different from the second state of the previous symbol. The second state of the symbol is identical to the first if the bit to be transmitted is logic 0. However, it is different if the bit is logic 1.

Figure C-6 shows that the ones in the original data end up with mid cell transitions in the bi-phase mark encoded data, while zeros in the original data do not. Note that the bi-phase mark encoded data always has a transition between bit boundaries.

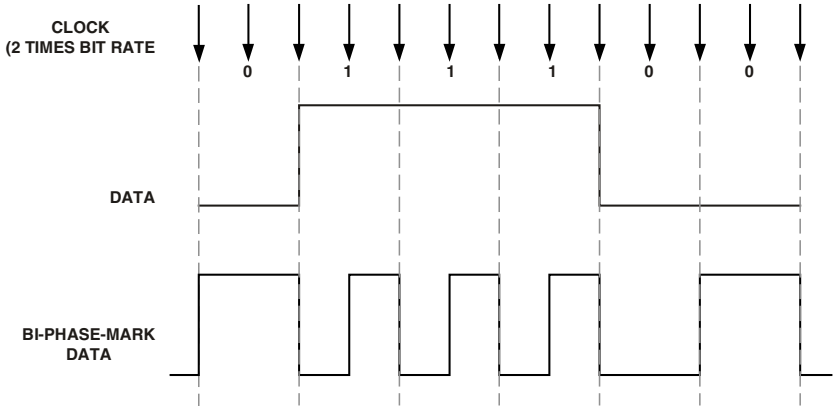


Figure C-6. Bi-phase Mark Encoding

Preambles

Preambles are specific patterns that provide synchronization and identify the subframes and blocks. To achieve synchronization within one sampling period and to make this process completely reliable, these patterns violate the bi-phase mark code rules, thereby avoiding the possibility of data imitating the preambles.

A set of three preambles, shown in Table C-2, are used. These preambles are transmitted in the time allocated to four time slots at the start of each subframe (time slots 0 to 3) and are represented by eight successive states. The first state of the preamble is always different from the second state of the previous symbol (representing the parity bit).

Table C-2. Preambles

Preamble	Preceding state 0	Preceding state 1	Description
X	11100010	00011101	Subframe 1
Y	11100100	00011011	Subframe 2
Z	11101000	00010111	Subframe 1 and block start

Like bi-phase code, the preambles are dc free and provide clock recovery. They differ in at least two states from any valid bi-phase sequence.

I INDEX

Numerics

128-channel TDM, 6-5
16-bit word lengths, 7-13
32-bit word lengths, 7-14
8-bit boot mode, 14-36
8-bit boot (SPI), 14-42
8-bit word lengths, 7-13

A

AAC compressed format, 11-18
AC-3 format, 11-18
access, bus, 2-32
accuracy (PWM), 10-21
active low frame sync select for frame sync (INVFSx) bit, 13-13
active low versus active high frame syncs, 6-30
active state multichannel receive frame sync select (LMFS) bit, 6-30
addressing
  boot modes, 14-35
  DMA controller, 2-18
  internal index, 2-18
  IOP, 2-18
  normal word (parallel port), 4-2
  SRAM memory, 4-8
address latch cycle, 4-5
address pins, parallel port, 4-8
ALE (address latch enable) cycle, parallel port, 4-4 to 4-23
ALE (address latch enable) equations, 4-23
arbitration, bus, 2-34
audience, intended, xxxi
audio
  bipahse encoded in S/PDIF, 11-2
  formats, IDP, 8-4
  formats, S/PDIF, 11-5, 11-13
  I^2S, SPORTs, 6-38
  lock error bit, S/PDIF, 11-15
  non-linear data, S/PDIF, 11-18
  no stream error bit, S/PDIF, 11-16
  parity error bit, S/PDIF, 11-16
  PCM data, S/PDIF, 11-15
  power control, PWM, 10-2
  restriction with SPORTs, 6-26
  transmission standards, SPORTs, C-3
audio data output (DIR_DAT_O) register, S/PDIF, 11-12
audio formats, C-1 to C-9
automotive products, 1-5

B

baud rate, 14-39
  setting, 7-34, 7-39
  SPIBAUD (serial peripheral interface baud rate) register, A-20
BHD (buffer hang disable) bit, 2-10, 2-33, 6-12, 8-25, A-13, A-34, A-48
bidirectional connections through the signal routing unit, 5-6
bidirectional functions (transmit, receive), 6-2

biphase
encoded audio stream, 11-11, 11-13
routing data, 11-4
S/PDIF receiver data register (DIR_I), 11-13
bits *See* peripheral specific bits, bits by name or acronym
block diagram
IDP, 8-4
IDP channel 0, 8-10
parallel port, 4-3
PLL, 14-5
PWM, 10-3
S/PDIF transmitter, 11-5
SPI, 7-6
SPORTs, 6-3
SRC, 12-5
system, processor, 1-3
boolean operator
OR, 8-30, 11-21
booting
boot kernal, 14-49
bootstrap loading, 14-32
DMA use in, 2-2
hardware use, 14-32
IVT addresses, 14-35
process, 14-32
SPI master mode, 14-38
SPI packing, 14-42
SPI slave mode, 14-41
boot memory select pin ($\overline{\text{BMS}}$) not used, 4-3
buffer
addressing, 2-18
data, 2-9
DMA count, 2-4
interrupts, 2-27, 4-15
parallel port operation, 4-11
SPORT data, 6-1
stalls, core, 2-33, 4-17
buffer *(continued)*
TCB allocation, 2-21
buffer enable (DIT_CHANBUF) bit (S/PDIF), 11-10
buffer hang disable (BHD) bit, 6-55, A-34, A-38, A-41
buses
access through I/O processor, 2-34
ALE cycles and, 4-20
contention, 7-37
contention in SPORTs, 6-43
determining parallel port cycles, 4-17
external, 4-16, 4-25, A-13
external parallel, 4-12
granting, 7-24
hold cycle enable (PPBHC) bit, 4-17
I^2S and, C-3
I/O address (IOA), 2-18
I/O data (IOD), 2-3, 2-14, 7-37
I/O processor (IOP), 2-3, 2-14, 6-21
packing sequence, 4-6
parallel port, 4-1
parallel port bus hold cycle enable (PPBHC) bit, 4-8
parallel port bus hold cycle enable (PPBHC) bits, A-12
parallel port bus status (PPBS) bit, 4-16, A-13
parallel port pins, 4-8
unpacking sequence, 4-7
bypass as a one-shot (strobe pulse), 13-14

C

capacitors, bypass, decoupling, 14-29
center-aligned paired PWM
double-update mode, 10-16
single-update mode, 10-14
chain assignment, I/O processor, 2-22

chained DMA, 2-3, 2-20, 2-24, 2-25
  chained DMA enable (SCHEN_A and SCHEN_B) bit, A-33
  chained DMA sequences, 2-21
  chain pointer (CPSPI) registers, SPI, 7-37, 7-39
  chain pointer (CPSPx) registers, SPORTs, 2-11, 6-51
  chain pointer (CPx) registers, 2-8, 2-21
  chain pointer registers (general), 2-21
  enable (SCHEN_A and SCHEN_B) bit, 6-51, A-41
  enable (SCHEN_A and SCHEN_B) bits, A-37
  parallel port, 4-19
  SPI, 2-31
  SPI chained DMA enable (SPICHEN) bit, 7-39
  SPORTs, 2-30, 6-51, A-33, A-37, A-41
chain insertion mode, SPORT, 2-25, 6-52
chain pointer registers, 2-21
changing SPI configuration, 7-26
channel
  buffer, 2-27
  defined, 2-2
  DMA, 2-3
  interrupt, 2-28
  priority scheme, 2-4
  status, 2-27, 2-29
channel B transmit status register (SPDIF_TX_CHSTB), A-72
channel selection registers, 6-46
chip select pin (CS), 4-3
clock A source (CLKASOURCE) bit, A-67
clock input (CLKIN) pin, 13-3, 13-21, 14-5
clocks and system clocking, 14-4 to 14-10
  bypass clock, 14-10
  clock and frame sync frequencies (DIVx) registers, 6-25
clocks and system clocking *(continued)*
  clock distribution, 14-28
  clock input (CLKIN) pin, 14-5
  clock polarity (CLKPL) bit, A-16
  clock rising edge select (CKRE) bit, A-32, A-40
  core clock, 14-9
  disabling the clock, 14-9
  frame sync bypass mode, 13-8
  frame sync bypass mode, direct bypass, 13-8
  frame sync bypass mode, one shot, 13-8
  hardware control, 14-7
  internal clock select (ICLK) bit, A-32, A-37, A-40
  managing for power savings, 14-13
  master clock (MCLK), 12-5, 13-19
  multiplexing, 5-15
  output divider, 14-8
  parallel port duration (PPDUR) bits, A-12
  peripheral clock (PCLK), 2-32, 4-4, 4-21, 6-2, 14-10
  PLL design, 14-4
  PLL input clock, 14-10
  precision clock generator registers, 13-9
  selecting clock ratios, 14-7
  software control, 14-8
  source select (MSTR) bit, A-32, A-37, A-40
  SPI clock phase select (CPHASE) bit, A-16
  SPORTs, 6-25 to 6-28
  VCO encodings, 14-9
  VCO frequency, 14-4
compand data in place, 6-5
companding (compressing/expanding), 6-5
conditioning input signals, 14-2
configuring frame sync signals, 6-9
connecting peripherals through DAI, 5-13

connections, DAI, 5-20
conventions, manual, xlii
converters, A/D and D/A, 4-10
core PLL, 13-2
core transmit/receive operations, 7-22
count (CSPx) DMA registers, 2-7
count (CSPx) registers, 2-20
count (IDP_DMA_Cx) registers, 8-20, 8-21
CPSPx (chain pointer) registers, 2-8
CRAT (PLL clock ratio bit), A-8
crosstalk, reducing, 14-29
CSPx (peripheral DMA counter) registers, 2-7, 2-20
customer support, -xxxvi

D

DAI
See also SRU
cframe sync routing control registers (Group C), A-89
clock routing control registers (Group A), A-80
clock routing control registers (group A), 5-20
configuration macro, 5-32
connecting peripherals with, 5-13
DAI interrupt falling edge (DAI_IRPTL_FE) register, 8-27
DAI interrupt rising edge (DAI_IRPTL_RE) register, 8-27
DAI_IRPTL_FE register
as replacement to IMASK, 5-27
DAI_IRPTL_H register, 8-23
DAI_IRPTL_H register as replacement to IRPTL, 5-26
DAI_IRPTL_L register as replacement to IRPTL, 5-26
DAI_IRPTL_PRI register, 5-26, 8-27
DAI_PIN_PULLUP register, A-106
DAI *(continued)*
DAI_PIN_STAT register, A-106
DAI_STAT register, 8-22, A-54, A-56, A-79
edge-related interrupts, 5-28
interrupt controller, 5-24 to 5-28
interrupt controller registers, A-77
interrupts, 5-25
miscellaneous signal routing registers (Group E), A-98
pin buffer enable registers (Group F), A-102
ping-pong DMA status (SRU_PINGx_STAT) register, A-55, A-56, A-80
pin signal assignment registers (Group D), A-93
pin status (DAI_PIN_STAT) register, A-106
resistor pullup enable (DAI_PIN_PULLUP) register, A-106
rules for routing, 5-16
serial data registers (Group B), A-85
SPORT SRU signal connections, 6-6
status (DAI_STAT) register, A-54, A-56, A-79
system configuration, sample, 5-31
system design, 5-3
use in, 5-20
DAI_IRPTL_RE register
as replacement to IMASK, 5-27
data
16-bit transfer mode, 4-9
8-bit transfer mode, 4-8
buffers in DMA registers, 2-11
direction control (SPTRAN) bit, A-34, A-38
packing and unpacking, 6-21
packing modes, 4-2

data cycle duration (PPDUR) bit, 4-8, A-12
data direction control (SPTRAN) bit, A-34, A-38
data-independent frame sync, 6-32
  (DIFS) mode, 6-32
data type
  and companding, 6-23
  and formatting (non-multichannel), 6-11, 6-22
  select (DTYPE) bit, A-32, A-40
data words
  in multichannel mode, 6-42
  packing, 6-21
  single word transfers, 6-47
  transferring, 4-23, 6-29, 6-45, 7-18
dead time equation, 10-13
debug, 14-2, 14-28
  DAI use in, 5-10, 5-29
  data buffer use in, 2-10
  registers, 5-29, 8-25, A-23
  SPI, 7-32
  SPORT, 6-54, A-44
  SPORTs, 6-24
  tools use in, 1-10
DERR_A (SPORT channel A error status) bit, A-35, A-39
DERR_B (channel B error status) bit, A-35, A-38
DIFS (data independent frame sync select) bit, A-33, A-37
digital applications interface, *See* DAI
digital loopback mode, *See* serial ports, loopback mode
DIR_NOAUDIO bit flag (non-PCM), 11-18
DIVEN (PLL divider enable) bit, 14-14, A-6, A-8
divisor (DIVx) registers, SPORT, 6-9, 6-25
DIVx (divisor) registers, SPORT, 6-9, 6-25, A-46
DMA
  booting, 2-2
  chained, 2-20, 2-24, 2-25
  chained, SPI, 2-31
  chained, SPORTs, 2-30
  chaining, 2-3
  channel, buffer registers, listed, 2-11
  channel, parameter registers, 2-11
  channel allocation, 2-10, 2-11
  channel disable, 2-20
  channel priority, 2-17
  channels, 2-2
  configuring in the I/O processor, 2-4
  controller enhancements, 1-11
  control registers, 2-11, A-10
  data buffer, 2-11, 2-21
  IDP ping-pong, 8-28
  IDP standard, 8-27
  interrupt-driven, 2-27
  interrupt regeneration, 2-33
  latency, 2-29
  memory-to-memory port, 3-3
  operation, master mode, 7-36
  operation, slave mode, 7-39
  parallel port, 4-12 to 4-14
  ping-pong, 2-3
  ping-pong enable (IDP_PING) bits, A-50
  ping-pong (IDP), 8-20, 8-28
  restrictions with parallel port, 4-14
  restriction using zero, 2-20
  sequence complete interrupt, 2-28
  SPI slave mode, 7-35, 7-36
  standard (non chained), 2-2
  starting and stopping, 2-25
  status (PPDS) bit, A-13
  switching from receiveto transmit mode, 7-42

DMA *(continued)*
switching from transmit to receive mode, 7-41
TCB memory allocation, 2-20
TCBs and, 2-20 to 2-31
transfers, 8-20
transmit or receive operations (SPI), 7-37
Dolby, DTS audio standards (S/PDIF), 11-10
double update mode (PWM), 10-20
DSP, architectural overview, 1-6
DSxEN (SPI device select) bits, 7-36, A-22
DTS format, 11-18
DTYPE (data type) bits, A-32, A-40
duty cycles and dead time in PWM, 10-13
DXS_B, DSX_A (data buffer channel A/B status) bit, A-35, A-38, A-39, A-42

E

early vs. late frame syncs, 6-31
ECPP (parallel port DMA external word count) register, 4-12, 14-38
edge-related interrupts, 5-28
EIPPx (parallel port DMA external address registers), 14-38
EIPPx (parallel port DMA external index) registers, 4-10, 4-14
EMPP (parallel port DMA external address modifier) register, 14-38
EMPPx (parallel port DMA external modify) registers, 4-14
enable
buffer, SPORTs, 6-10
buffers, SRU output, 7-17
channels, SPORTs, 6-19
companding, SPORTs, 6-24
DMA interrupt (INTEN) bit, 7-29
duplex, SPI, 7-3
error, multi master detect (SPI), 7-19
EXT_CLK mode, 9-17
enable *(continued)*
external cock synchronization, PCG, 13-9
frame sync, PCG, 13-7
full duplex, SPORTs, 6-4
I^2S mode, SPORTs, 6-39
interrupts, memory-to-memory, 3-3
left-justified mode, SPORTs, 6-37
multichannel, SPORTs, 6-12
packing, SPI, 7-18
PCGs, 13-5, 13-10
peripheral timer, 9-7
pin buffer, timer, 9-4
PLL, S/PDIF, 11-15
pulse width modulation groups, 10-10
PWM_OUT mode, 9-9
PWM output signals, 10-4, 10-5
serial mode, SPORTs, 6-34
slave, SPI, 7-3
SPDIF transmit buffer, 11-10
SPI, 7-17
SPI DMA, 7-37
$\overline{\text{SPIDS}}$ (ISSEN) bit, 7-19
SPI Rx, Tx, 7-9
SPI slave, 7-9, 7-24
standalone mode, S/PDIF, 11-9
synchronize (counter) bits, PWM, 10-7
WDTH_CAP mode, 9-14
endian format, 6-21, 7-2, A-32
equation
address latch (ALE), 4-23
address latch cycles, 4-22
dead time, 10-13
duty cycles in PWM, 10-15
frame sync frequency, 6-26
frame sync pulse (SPORT), 6-26
parallel port access (8, 16-bit), 4-21
peripheral timer period, 9-13
pulse width modulation switching frequency, 10-11

equation *(continued)*
PWM dead time, 10-13
serial clock frequency, 6-25
serial port clock divisor, 6-25
SPI clock baud rate, A-20
SRAM access, parallel port, 4-22
error status bits, SPI, 7-21
examples
chained DMA, 4-26
DAI clock, 5-15
interrupt latency, regeneration, 2-33
PCG channel output enable, 13-21 to 13-25
PCG initialization, 13-19
PCG setup for I2S or left-justified DAI, 13-16
power management, 14-13 to 14-15
programming SPORTs, 6-57
programming the IDP, 8-31, 8-33
programming the parallel port, 4-26
programming the PWM, 10-24
programming the SPI port, 7-44
examples, timing
IDP I^2S, 8-8
IDP left-justified, 8-8
SPI clock, 7-32
SPI transfer protocol, 7-15
SPORT framed vs. unframed data, 6-32
SPORT normal vs. alternate framing, 6-32
Expert DAI software, 5-32
EXT_CLK (external event watchdog) mode, 9-7
external device or memory
reading from, 4-6
writing to, 4-7
external event watchdog (EXT_CLK) mode, 9-2, 9-17
external frame sync routing (DIR_LRCLK_FB_O, 11-12
external Index register, 2-30, 2-31, 4-14
external word count (ECPP) register, 4-10

F

FE, format extension, *See* serial ports, word length
FIFO
see also buffer
data packing in IDP, 8-7
flush, memory-to-memory, 3-3
IDP, 8-4
IDP modes use, 8-13
overflow, IDP, 8-9
receive, SPORT, 6-15
SPI, 7-7
SPI DMA, 7-23, 7-30, 7-37
SPI DMA programming, 7-44
status bit, A-13
to memory data transfer, 8-18
transmit, SPORT, 6-13
transmit status, 4-11
FIG, frame ignore, *See* serial ports, framed and unframed data
FLAG pin multiplexing (core), 14-18
flags
contention in SPI, 7-19
core pin use as, 14-18
DAI routing of, 5-20
error, 7-20
flag interrupt mode (IRQxEN) bits, A-4
input/output (FLAGx) pins, 6-10, 7-7, 14-18
multi master SPI systems, 7-9
restriction with SPI bits, 14-24
slave select signals, 7-36
SPI slave operation, 7-4
SPORT pins, 6-10
FLAGx pins, 6-10, 7-7, 14-18
framed versus unframed data, 6-28

frame sync
active low vs. active high, 6-30
A source (FSASOURCE) bit, A-67
delay, SPORT (MFD), 6-45
early vs. late, 6-31
equations, 13-13
frequencies, 6-25
in multichannel mode, 6-42
internal vs. external, 6-29
options (FS_BOTH), 6-33
options (FS_BOTH and DIFS), 6-33
output, synchronizing, 13-9
PCG B source (FSBSOURCE) bit, 13-15
rates, setting, 6-36
required (SPORT FSR) bit, A-32
routing control (SRU_FS0) registers (group C), A-89
signals, configuring, 6-9
framing bits, 6-37, 6-39
frequency of the frame sync output, 13-10
FRFS (frame on rising frame sync) bit, 6-37, 6-39
FS_BOTH (frame sync both) bit, A-34
FSM, frame synchronization mode, *See* serial ports, framed and unframed data
FSP, frame synchronization polarity, *See* serial ports, framed and unframed data
FSR (frame sync required) bit, A-32
full-duplex operation, specifications, 6-9

G

generators, optional reset, 14-27
GM (get more data) bit, 7-20, 7-24, 7-34, A-16
ground plane, 14-29
ground plane, in PCB design, 14-29
group descriptions, signal routing unit, 5-13
group routing, 5-14

H

hang, buffer enable bit, 14-37
hang, core, 2-33, 6-14, 6-15, 6-17, 6-48, 7-22, 7-44, 8-25, A-13, A-34
hang, DMA controller, 4-12, 4-14
hardware interrupt signals $\overline{\text{IRQ2-0}}$, 14-18
hold cycles, parallel port, 4-8
hold time
external latch, 4-4
inputs, 14-4
recognition of asynchronous input, 14-4
WR signal, 4-5
hysteresis on $\overline{\text{RESET}}$ pin, 14-2

I

I^2S
example for DAI, 13-16
transmit and receive channel order (FRFS), 6-37, 6-39
(Tx/Rx on left channel first), 6-19
(Tx/Rx on right channel first), 6-19
ICLK (internal clock select) bit, A-32, A-37, A-40
ICPP (DMA internal word count) register, 4-12, 14-38
IDP
buffer, 2-9
channel 0 diagram, 8-10
control (IDP_CTL0) register, 8-27, A-47
control (IDP_CTL1) register, 8-27, A-49, A-50
(DAI) interrupt service routine steps, 8-23
DMA, ping-pong, 8-28
DMA, standard, 8-27
DMA control registers, 2-6, 2-7
DMA count (IDP_DMA_Cx) register, 8-20, 8-21

IDP *(continued)*
DMA index (IDP_DMA_Ix) register, 8-20, 8-21
DMA modify (IDP_DMA_Mx) register, 8-20, 8-21
FIFO (IDP_FIFO) register, 8-17, 8-18, 8-23, A-53
FIFO memory data transfer, 8-18
FIFO register (IDP_FIFO register), 8-17, A-53
illustrated, 8-2
interrupt driven transfers, 8-18
interrupts, 8-18, 8-27
memory data transfer, 8-18
multiplexed channels, 8-4
multiplexing, 8-4, 8-9
multiplexing register, 14-18
overflow, FIFO, 8-9
packing modes, 8-13, 8-14
PDAP control (IDP_PDAP_CTL) register, A-50
PDAP control (IDP_PP_CTL) register, A-50
ping-pong DMA, 8-20
pin status register, A-106
polarity of left-right encoding, 8-8
programming examples, 8-31
right-justified mode, 8-26
serial inputs, 8-4
throughput, 8-6
IDP bits
buffer hang disable (IDP_BHD), 8-25
bus hang disable (IDP_BHD), A-48
clear buffer overflow (IDP_CLROVR), A-48
data format (IDP_SMODEx), A-49
DMA enable (IDP_DMA_EN), 8-27, A-48
DMA status (IDP_DMAx_STAT), 8-22, 8-23, A-55, A-80
IDP bits *(continued)*
enable (IDP_ENABLE), 8-27, 8-28, A-48
FIFO number of samples (IDP_FIFOSZ), A-55, A-80
FIFO samples exceed interrupt (IDP_FIFO_GTN_INT), 8-27
frame sync format (IDP_SMODEx), 8-6, 8-26, A-53
IDP_DMA_EN bit
do not set, 8-19
monitor number of samples (IDP_NSET), A-48
PDAP clock edge (IDP_PDAP_CLKEDGE), 8-26, A-52
PDAP enable (IDP_PDAP_EN), 8-28, A-52
PDAP input mask bits, 8-26
PDAP packing mode (IDP_PDAP_PACKING), A-52
PDAP reset (IDP_PDAP_RESET), A-52
ping-pong DMA enable (IDP_PING) bits, A-50
port select (IDP_PORT_SELECT), A-51
port select (IDP_PP_SELECT), 8-26
reset (IDP_PDAP_RESET) bit, A-52
IDP_CTL0 (input data port control) register, 8-27, A-47
IDP_CTL1 (input data port control) register, 8-27, A-49, A-50
IFS (internal frame sync select) bit, A-33, A-41
IISPx (serial port DMA internal index) registers, 2-6, 2-18
IMPP (parallel port DMA modify) register, 14-38

IMSPI (serial peripheral interface address modify) register, 7-37, 7-39
IMSPx (SPORT DMA address modifier) registers, 2-6, 2-18
INDIV (input divisor) bit, A-8
input setup and hold time, 14-4
input signal conditioning, 14-2
input slave select enable (ISSEN) bit, 7-19, A-16
input synchronization delay, 14-18
instructions
  atomic, using for clock and frame sync, 13-11, 13-12
INTEN (DMA interrupt enable) bit, 7-29
interconnections, master-slave, 7-3
interface to core orinternal DMA viaRXPP register, 4-6
internal index register, IOP, 2-20, 2-21, 2-24, 2-30
internal memory
  DMA index (IDP_DMA_Ix) registers, 8-20, 8-21
  DMA index (IISPx) registers, 2-6, 2-18
  DMA modifier (IDP_DMA_Mx) registers, 8-20, 8-21
  DMA modifier (IMSPx) registers, 2-6, 2-18
internal serial clock, setting, 6-36
internal vs. external frame syncs, 6-29
INTERR (enable interrupt on error) bit, 7-29
interrupt controller, digital applications interface, 5-24
interrupt driven DMA, I/O processor,2-27
interrupt latch/mask (LIRPTL) registers, 7-29
interrupts
  conditions for generating interrupts, 6-50
  digital applications interface, 5-24
interrupts *(continued)*
  parallel port, 4-19
  peripheral timers, 9-20
  priority interrupt control registers (PICR), B-1
  programmable, B-1 to B-5
  SPI mask, 7-29
  SRC, 12-19
  timer pins, 14-18
  vector, sharing, 6-52
interrupt vector table. *See* IVT
INVFSx (active low frame sync select for frame sync) bits, 13-13
I/O interface to peripheral devices, 6-1
IOP register set, 6-10
I/O processor
  address bus (IOA), 2-18
  and addressing, 2-18
  baud rate, 7-39
  buffer
    DMA data, 2-21
    DMA data, listed, 2-11
  bus access, 2-32
  bus arbitration, 2-34
  chain assignment, 2-22
  chained DMA, 2-21
  chain pointer (CPSPI) register, 2-21
  chain pointer registers, 2-8
  configuring DMA, 2-4
  core-interrupt I/O, 2-28
  count registers, 2-7, 2-20
  DAI interrupt registers (DAI_IRPTL_H, DAI_IRPTL_L), 2-27
  data buffers in DMA, 2-11
  DMA channel priority, 2-17
  DMA channel registers, 2-11
  DMA interrupt registers, 2-27
  DMA sequence complete interrupt, 2-28
  IDP buffer, 2-9
  latency, 2-29, 2-33

I/O processor *(continued)*
memory access, DMA, 2-27
non DMA interrupts, 2-28
program control interrupt (PCI) bit, 2-22, 2-24, 2-27, 2-28
regenerated interrupts, avoiding, 2-33
stall conditions, 2-32
standard (non chained) DMA, 2-2
TCB memory allocation, 2-20
transfer types, 2-1
IRQ2-0 (hardware interrupt) pins, 14-18
ISSS (input service select) bit, A-22
IVT
addresses, 14-32, 14-35
boot kernal use, 14-49

J

JTAG
interface pins, 14-17

K

kernel boot timing, 14-47

L

LAFS (late transmit frame sync select) bit, 6-19, 6-31, 6-36, A-33, A-37
latchup, 14-2
latency
input synchronization, 14-18
in SPORT registers, 6-55
parallel port control register (PPCTL), 4-23
left-justified mode, 6-36, C-5
control bits, 6-37
IDP, 8-4, 8-19, A-49
IDP FIFO, 8-24
IDP frame sync, 8-8
PCGs, 13-16, 13-18
left-justified mode *(continued)*
PWM, 10-7, A-25
S/PDIF, 11-3, 11-5, 11-7, A-71
SPORTs, setting, A-37
SRC, 12-2, 12-8, 12-12, A-61
SRC timing, 12-13
Tx/Rx on FS falling edge, 6-19
Tx/Rx on FS rising edge, 6-19
LIRPTL (interrupt) registers, 7-29
loopback mode
setting, SPI, A-17
setting, SPORTs, A-44
S/PDIF, 11-21
SPI, 7-33
SPORTs, 6-54
timers, 9-21
low active transmit frame sync (INVFSB) bit, A-68
LRFS (SPORT logic level) bit, 6-30
LSBF (least significant bit first) bit, A-32

M

making connections via the signal routing unit, 5-16
manual
contents, xxxii
conventions, xlii
new in this edition, xxxiv
related documents, xxxix
manual revisions, xxxiv
maskable interrupts, SPI, 7-29
master input slave output (MISOx) pins, 7-5, 7-14
slave output, 7-14
master mode
DAI, 5-3, 5-21
I/O processor, 2-14
SPI, 7-33
SPORTs, 6-6, 6-7, 6-37, 6-39, 6-42, 6-44

master mode *(continued)*
SPORTs enable, 6-34, 6-39, 6-44
master mode operation
SPORTs, 6-30
master out slave in (MOSIx) pin, 7-5, 7-14
master-slave interconnections, 7-3
MCM, multichannel mode, *See* serial port modes, multichannel mode
memory
data transfer, FIFO, 8-18
mapped IOP (RXSPI and TXSPI) buffer registers, 7-29
memory-mapped registers, A-2
parallel port access, 4-22
TCB allocation for DMA, 2-20
memory-mapped peripheral, 4-21
memory-mapped registers, 2-5, 2-32, 4-11, 5-2
memory-to-memory port
buffers, 3-3
channels, DMA, 3-2
features, 3-2
FIFO, 3-3
interrupts, 3-3
programming, 3-4
throughput, 3-3
memory transfer types, 2-1
MISCAx_I (signal routing unit external miscellaneous) register, 13-14
miscellaneous signal routing (SRU_EXT_MISCx) registers (Group E), A-98
MISOx pins, 7-5, 7-14
mode
16-bit, 4-5, 4-9, 4-10
8-bit, 4-5, 4-8
left-justified (IDP), 8-4
left-justified (SPORT), 6-36, C-5
master (SPI), 7-44
open drain (SPI), 7-17
mode *(continued)*
right-justified (IDP), 8-4
serial mode settings (IDP), 8-6
single channel double frequency (SPDIF), 11-8
standard serial, 6-34, C-2
TDM (SPORT), 6-5
timer, A-56
two channel (SPDIF), 11-8
mode fault error (MME) bit, 7-19
mode fault (multimaster error) SPI DMA status (MME) bit, 7-19
modes
audio, C-2 to C-9
MOSIx pins, 7-5, 7-14
most significant byte first (MSBF) bit,A-16
MPEG-2 format, 11-18
MRxCCSx (SPORT receive compand) registers, A-46
MRxCSx (SPORT receive select) registers, A-46
MSBF (most significant byte first) bit,A-16
MTxCCSx (serial port transmit compand) registers, A-45
MTxCCSy and MRxCCSy (multichannel compand select) registers, 6-22
MTxCSx (serial port transmit select) registers, A-45
multichannel- A and B channels, 6-20
multichannel compand select (MTxCCSy and MRxCCSy) registers, 6-22
multichannel operation, 6-40
multiplexing, 14-24
FLAG use of, 14-19, 14-21
IDP channels, 8-4
parallel port, 4-4
PDAP use of, 14-19, 14-21
PDAP with SIP0, 8-9
pins, 14-17 to 14-24
PWM to parallel port, 14-24

multiplexing *(continued)*
SPORT data channel, 6-38, 6-49
SRU clock, 5-15
SRU signals, 5-3

N

normal frame sync, 6-31

O

one shot, defined, 13-14
one shot frame sync A or B (STROBEx) bits, 13-13
one shot option (STROBEB) bit, 13-14
OPD (SPI open drain mode), 7-17
OPMODE (serial port operation mode) bit, 6-34, 6-37, A-37
OR, logical, 8-30, 11-21
output pulse width, defined, 13-15
over-modulation, in PWM, 10-18

P

package availability, 1-5
packing
16 to 32-bit packing (PACK) bit, 6-11, 6-20, 6-21, 6-50, 6-53, A-32, A-37, A-40
modes in IDP_PP_CTL, illustrated, 8-13
sequence for 32-bit data (parallel port), 4-6
serial peripheral interface data, 7-14
serial peripheral interface (PACKEN) bit, A-17
serial port data, 6-21
parallel data acquisition port control (IDP_PP_CTL) register, A-50
parallel data acquisition port (PDAP), 8-9
parallel port
address latch enable (ALE) cycle, 4-5
bus cycles, determining, 4-17
bus status (PPBS) bit, 4-24
clearing registers, 4-18
configuring, 4-11
control (PPCTL) register, 4-5, 4-11, A-11
data packing, 4-2
data transfer, 16-bit mode, 4-9
data transfer, 8-bit mode, 4-8
data transfer, core stall driven, 4-17
data transfer, interrupt driven, 4-15
data transfer, known duration accesses, 4-16
data transfer, status driven, 4-16
DMA address (IMPP) register, 14-38
DMA enable (PPDEN) bit, 4-11, 4-12, 4-18, 4-24
DMA external address (EIPPx) registers, 4-14, 14-38
DMA external address (EMPP) register, 4-14
DMA external address modifier (EMPP) register, 14-38
DMA external word count (ECPP) register, 4-12, 14-38
DMA internal word count (ICPP) register, 4-12, 14-38
DMA start internal index address, 14-38
DMA use in, 4-12
external word count (ECPP) register, 4-10
hold cycle, 4-8
interrupt (PPI) signal, 4-19
latency in PPCTL register, 4-23
multiplexed pin functions, 4-4, 14-17, 14-24
multiplexing with PWM, 14-24
packing sequence for 32-bit data, 4-6

parallel port *(continued)*
pins, 4-3, 4-9
polarity, 4-5
read cycle, 4-5
registers, 4-10, A-10
restrictions in use, 4-13
signals, 4-3
SRAM memory, 4-8
stalls in, 4-17
system
configure and enable, A-11
TCB, 2-30
throughput, 4-19, 4-21, A-12
transfer protocol, 4-8
write cycle, 4-5
parallel port bits
ALE polarity level (PPALEPL), A-13
buffer hang disable (PPBHD), A-13
bus hold cycle enable (PPBHC), 4-8, 4-17, A-12
bus status (PPBS), 4-16, A-13
data cycle duration (PPDUR), 4-8, A-12
DMA enable (PPDEN), A-12
DMA status (PPDS) bit, A-13
enable (PPEN), A-12
external data width (PP16), A-12
FIFO status (PPS), A-13
hold cycle (PPBHC), 4-8
parallel port chaining status (PPCHS), A-14
PP16 (external data width), A-12
PPBHD (buffer hang disable), A-13
PPBS (bus status), A-13
PPDEN (DMA enable), A-12
PPDS (parallel port DMA status), A-13
PPEN (enable), A-12
PPI (parallel port interrupt), 4-16, 4-18
PPS (FIFO status), A-13
PPTRAN (transmit/receive select), A-12
parallel port bits *(continued)*
transmit/receive select (PPTRAN), 4-6, 4-24, 4-25
PCG
active low frame sync select for frame sync (INVFSx) bits, 13-13
bypass mode, 13-13
clock A source (CLKASOURCE) bit, A-67
clock input (CLKIN) pin, 13-3, 13-21
control (PCG_CTL_Ax) registers, 13-15, A-66
division ratios, 13-17
frame sync A source (FSASOURCE) bit, 13-15, A-67
frame sync B source (FSBSOURCE) bit, 13-15
frame syncs, 13-13
frequency of the frame sync output, 13-10
one shot frame sync A or B (STROBEx) bits, 13-13
one shot option, 13-14
PCG_CTLA0 (control) register, A-66
phase shift of frame sync, 13-11
pulse width (PCG_PW) register, 13-13, 13-14
setup for I^2S or left-justified DAI example, 13-16
synchronization with the external clock, 13-9
PCI (program control interrupt) bit, 2-22, 2-23, 2-27, 2-28, 4-13
parallel port, 4-14, 4-19
SPI, 7-12
SPORTs, 6-52
PCM audio, 11-18
PDAP
enable (IDP_PDAP_EN) bit, A-52

PDAP *(continued)*
port mask bits (IDP_Pxx_PDAPMASK), A-51
(rising or falling) clock edge (IDP_PDAP_CLKEDGE) bit, A-52
PDAP control (IDP_PDAP_CTL) register, A-50
peripheral devices, I/O interface to, 6-1
peripheral interrupt priority control (PICR) registers, B-1
peripherals, overview, 1-7
peripheral timers
configuring, 9-5, A-56
external event watchdog (EXT_CLK) mode, 9-6, 9-17
input/output (TMRx) pin, 9-4
interrupts, 9-20
invalid conditions, 9-11
modes, 9-4
period, configuring, 9-6
period equation, 9-13
pulse width count and capture (WDTH_CAP) mode, 9-14
pulse width modulation (PWMOUT) mode, 9-9
rectangular signals, 9-12
RTI instruction, 9-20
single pulse generation, 9-13
TIMERx pin, 9-5
TMRPDN (peripheral timer enable/disable) bit, A-9
watchdog, 9-20
word count (TMxCNT) registers, 9-5
peripheral timers registers, 9-7, A-56
high word period (TMxPRD) registers, 9-5
high word pulse width (TMxW) registers, 9-5
period (TMxPRD) registers, 9-6
pulse width (TMxW) registers, 9-6
peripheral timers registers *(continued)*
timer control (TMxCTL), 9-5, A-56
timer count (TMxCNT), 9-5, 9-6
timer global status and control (TMSTAT), 9-7
timer status (TMxSTAT), A-57
timer width (TMxW), 9-6
timer word period (TMxPRD), 9-6
word count (TMxCNT) registers, 9-6
phase shift of frame sync, 13-11
PICR (peripheral interrupt priority) registers, B-1
ping-pong DMA, 2-3, 8-20, 8-28
pins
See also signals
address connection (SDRAM A15-0), 4-8
data (SDRAM D7-0), 4-8
descriptions, 14-16
FLAGx, 6-10
multiplexed, 14-17
open drain output, 7-17
parallel port, 4-3
parallel port bus, 4-8
$\overline{\text{RESET}}$, 14-2
test clock (TCK), 14-16
test data input (TDI), 14-16
test data output (TDO), 14-16
test mode select (TMS), 14-16
test reset ($\overline{\text{TRST}}$), 14-16
timer (through SRU), 9-4
write($\overline{\text{WR}}$), 4-9
plane, ground, 14-29
PLL
block diagram, 14-5
function, 14-5
PLLBP (PLL bypass bit), 14-10, 14-14, A-8
PLLDx (PLL divider) bits, A-8
PLLM (PLL multiplier) bit, A-8

PMCTL (power management control) register, A-6, A-8
PMCTL register use in multiplexing, 14-18
polarity
  ALE in parallel port, 4-5
  IDP left-right encoding, 8-8
  PWM double-update mode, 10-16
  PWM signals, 10-20
  PWM single update mode, 10-14
  setting in parallel port (PPALEPL bit), A-13
  SPDIF connections, C-14
  SPI clock, 7-14, 7-26
porting from previous SHARCs
  symbol changes, 1-11
power management
  control register (PMCTL), A-6
  examples, 14-13 to 14-15
  PMCTL (power management control) register, 14-5, 14-8, 14-10
  saving, 14-9
  savings, 14-10
power savings, 14-13
power supply, monitor and reset generator, 14-27
power-up reset circuit, 14-27
power-up *See system design*
PP16 (parallel port external data width) bit, A-12
PPALEPL (parallel port ALE polarity level) bit, A-13
PPBHC (parallel port bus hold cycle) bit, 4-8, A-12
PPBHD (parallel port buffer hang disable) bit, A-13
PPBS (parallel port bus status) bit, A-13
PPCTL (parallel port control) register, 4-5, 4-11, A-11, A-12
PPDEN (parallel port DMA enable) bit, 4-11, 4-12, 4-18, 4-24, A-12
PPDS (parallel port DMA status) bit, A-13
PPDUR (parallel port data cycle duration) bit, 4-8, A-12
PPEN (parallel port enable) bit, 4-5, A-12
PPI (parallel port interrupt) bit, 4-16, 4-18
PPI (parallel port interrupt) signal, 4-19
PPPDN (parallel port clock enable) bit, A-9
PPS (parallel port FIFO status) bit, A-13
PPTRAN (parallel port transmit/receive select) bit, 4-6, 4-24, 4-25, A-12
preambles, S/PDIF, C-15
precision clock generators. *See* PCG
printed circuit board design, 14-29
priority, channel, 2-17
processor core, overview, 1-7
product details, 1-5
product-related documents, xxxviii
program control interrupt (PCI) bit, 2-22, 2-24, 2-27, 2-28, 6-52
programmable clock cycles, 4-8
programmable interrupt registers (PICRx), B-1 to B-5
programming examples
  input data port, 8-31
  PCG channel output enable, 13-21 to 13-25
  PCG initialization, 13-19
  precision clock generators, 13-23 to 13-25
  serial ports, 6-57 to 6-65
  SPORTs, 6-57
pulse, clock, in serial ports, 6-9
pulse, frame sync delay in serial ports, 6-45
pulse, frame sync formula, 6-26
pulse code modulation (PCM), 6-38
pulse width count and capture (WDTH_CAP) mode, 9-14
pulse width modulation (PWMOUT) mode, 9-9

PWM
16-bit read/write duty cycle registers, 10-13
accuracy in, 10-21
block diagram, 10-3
center-aligned paired PWM double-update mode, 10-16
channel duty control (PWMA, PWMB) registers, A-29
crossover mode, 10-23
dead time equation, 10-13
duty cycles, 10-13
equations, 10-14 to 10-17
global control (PWMGCTL) register, A-23
global status (PWMGSTAT) register, A-24
multiplexing with parallel port, 14-24
multiplex with parallel port, 14-24
over-modulation, 10-18
polarity of signals, 10-20
switching frequency equation, 10-11
update modes, 10-20
PWM bits
crossover (PWM_AXOV, PWM_BXOV), 10-23
PWMGCTL (pulse width modulation global control) register, A-23
PWMGSTAT (pulse width modulation global status) register, A-24
PWMOUT (pulse width modulation) mode, 9-9

R

read cycle, 4-5
receive busy (overflow error) SPI DMA status (SPIOVF) bit, A-19
receive busy (overflow error) SPI status (ROVF) bit, 7-20
receive data, serial port (RXSPx) registers, 2-9
receive data, SPI (RXSPI) register, 7-20
receive data buffer shadow (RXSPI_SHADOW) register, A-23
receive data (RXSPI) buffer, 7-7
receive overflow error (SPIOVF) bit, 7-29, 7-30, 7-31
receive shift (RXSR) register, 7-5
reception error bit (ROVF), 7-20
register drawings, reading, A-3
register writes and effect latency, SPORTs, 6-13
related documents, xxxix
reset
generators, 14-26
RESET pin, 14-2
input hysteresis, 14-2
resolution (PWM), 10-21
restrictions
parallel port, 4-13
right justified mode, C-5
right-justified mode
IDP, 8-2, 8-4, 8-26
S/PDIF, 11-5, 11-6, 11-7, A-71
SPORTs, 6-13, 6-20
SRC, 12-12, 12-13
SRC, timing, 12-13
ROVF_B or TUVF_B (channel B error status) bit, A-42
ROVF (receiption error) bit, 7-20
RS-232 device
restrictions, 6-10
RXFLSH (flush receive buffer) bit, 7-31, 7-42, 7-43
RXS_A (data buffer channel B status) bit, A-35, A-39, A-42
RXSPI, RXSPIB (SPI receive buffer) registers, 7-20, 7-22, 7-29

RXSPI_SHADOW, RXSPIB_SHADOW (SPI receive buffer shadow) registers, A-23
RXSPx (serial port receive buffer) registers, 2-9
RXSR (SPI receive shift) register, 7-5
RXS (SPI data buffer status) bit, A-21

S

saving power, 14-9, 14-13
SCHEN_A and SCHEN_B (serial port chaining enable) bit, A-33, A-37, A-41
SDEN (serial port DMA enable) bit, A-33, A-37, A-41
SENDZ (send zero) bit, A-15
SENDZ (send zeros) bit, 7-20, 7-34
serial clock (SPORTx_CLK) pins, 6-9
serial inputs, 8-4
serial peripheral interface, *See* SPI
setting up DMA on SPORT channels, 6-50
setup time, inputs, 14-4
shadow registers
  DAI, 5-29
  IDP, 8-25
  SPI, A-23
short word data, I/O processor, 2-18
signal routing unit external miscellaneous (MISCAx) registers, 13-14
signal routing unit *See* SRU, DAI
signal routing unit (SRU), 9-4
signals
  CLKIN, 14-5
  DAI routing, 5-20
  PWM waveform generation and, 9-12
  sensitivity in serial ports, 6-8
  serial port, 6-4, 6-9, 6-25
  slave select (SPI), 7-40
  SPORT, 6-6
  timer, 9-4
single channel double frequencey mode, 11-8
single update mode (PWM), 10-20
slave mode
  booting, 14-41
  booting SPI, 14-38
  DAI, 5-3, 5-21
  DAI pin use in, A-98, A-105
  DMA operations (SPI), 7-39
  IDP, 8-3
  SPI, 7-3, 7-7, 7-8, 7-10, 7-35, A-16, A-22
  SPORTs, 6-27, 6-30, 6-39, 6-56
software
  DAI plug-in, 5-32
  documentation, -xl
  interrupts, 5-24
  reset in SPI, 7-17
  ROM security, 1-9
  VisualDSP, xxxix
SP1PDN (SPORT1 clock enable) bit, A-9
SP2PDN (SPORT2 clock enable) bit, A-9
SP3PDN (SPORT3 clock enable) bit, A-9
SPCTLx control bit comparison in four SPORT operation modes, 6-11
SPCTLx control bits for left-justify mode, 6-35
SPCTLx (serial port control) registers, 6-9, 6-10
S/PDIF
  See also S/PDIF bits; S/PDIF registers
  AAC compressed format, 11-18
  AC-3 format, 11-18
  audio data output, 11-12
  audio standards, 11-10
  biphase encoding, 11-4
  block structure, C-11
  buffer enable (DIT_CHANBUF) bit, 11-10
  clock (SCLK) input, 11-4

S/PDIF *(continued)*
compressed audio data, 11-18
DTS format, 11-18
external PLL feedback point connection, 11-14, A-98
frame sync (LRCLK) input, 11-4
MPEG-2 format, 11-18
non-linear audio data, 11-18
output routing, 11-5
pin descriptions, receiver, 11-11
pin descriptions, transmitter, 8-5, 11-3
preambles, C-15
programming guidelines, 11-10
reference PLL point connection, 11-14, A-98
right-justified mode, 11-5, 11-6, 11-7
serial clock input, 11-10
serial data (SDATA) input, 11-4
single-channel, double-frequencey format, 11-8
subframe format, C-12
time division multiplexed (TDM) mode, 11-11
timing, 11-4
two channel mode, 11-8
S/PDIF bits
biphase error (DIR_BIPHASEERROR), A-76
buffer enable (DIT_CHANBUF), 11-10
channel status buffer enable (DIT_CHANBUF), A-71
channel status byte 0 A (DIT_B0CHANL), A-72
channel status byte 0 B (DIT_B0CHANR), A-72
channel status byte 0 for subframe A (DIR_B0CHANL), A-76
channel status byte 0 for subframe B (DIR_B0CHANR), A-76
disable PLL (DIR_PLLDIS), A-74
S/PDIF bits *(continued)*
frequency multiplier (DIT_FREQ), A-71
lock error (DIR_LOCK), A-74
lock receiver status (DIR_LOCK), A-76
mute receiver (DIR_MUTE), A-74
mute transmitter (DIT_MUTE), A-71
non-audio frame mode channel 1 and 2 (DIR_NOAUDIOLR), A-76
non-audio subframe mode channel 1 (DIR_NOAUDIOL), A-76
parity biphase error (DIR_BIPHASE), A-74
parity (DIR_PARITYERROR), A-76
select single channel double frequency mode channel (DIT_SCDF_LR), A-71
serial data input format (DIT_SMODEIN), A-71
single channel double frequency channel select (DIR_SCDF_LR), A-74
stream disconnected (DIR_NOSTREAM), A-76
transmit single channel double frequency enable (DIT_SCDF), A-71, A-74
transmitter enable (DIT_EN), A-71
validity bit A (DIT_VALIDL), A-71
validity bit B (DIT_VALIDR), A-71, A-72
validity (DIR_VALID), A-76
S/PDIF registers
audio data output (DIR_DAT_O) register, 11-12
channal A transmit status (SPDIF_TX_CHSTA), A-72
channal B transmit status (SPDIF_TX_CHSTB), A-72
external PLL feedback point connection (DIR_LRCLK_FB_O), 11-14

S/PDIF bits *(continued)*
frame sync output (SPDIF_PLLCLK_I), 11-12
left channel status for sub-frame A (DIRCHANL), A-77
receiver biphase encoded data input (DIR_I), 11-13
receiver frame sync output (DIR_FS_O), 11-12
receiver status (DIRSTAT), A-75
receiver TDM output (DIR_TDMCLK_O), 11-12
reference PLL point connection (DIR_LRCLK_REF_O), 11-14
SRU control, 11-5, 11-12
transmit control (DITCTL), 11-10, A-70
SPDIF_TX_CHSTA (Sony/Philips digital interface channel status) register, A-72, A-73
special IDP registers, A-78
SPEN_A (serial port channel A enable) bit, 6-37, 6-39, A-32, A-36
SPEN_B (serial port channel B enable) bit, 6-37, 6-39
SPI
See also SPI bits; SPI registers
block diagram, 7-5
booting, 14-38, 14-41
boot packing, 14-42
chained DMA, 2-31
chaining, DMA, 7-12, 7-23, 7-25, 7-29, 7-37, 7-39, 7-44
change clock polarity, 7-26
changing configuration, 7-26
clock phase, 7-15
clock (SPICLK) pin, 7-14
clock (SPICLK) signal, 7-5
configuring and enabling, 7-36
SPI *(continued)*
DMA, switching from transmit to receive mode, 7-41
enabling, 7-17
examples, programming the SPI port, 7-44
examples, timing, 7-32
examples, transfer protocol, 7-15
features, 7-2
functional description, 7-5
general operations, 7-5
initiate transfer, 7-11, 7-12
interconnections, master-slave, 7-3
interface signals, 7-3
interrupt, 7-23, 7-28, 7-38
loopback mode, 7-33
master boot mode, 14-38
master input slave output (MISOx) pins, 7-5
master mode, 7-33
master mode operation, configuring for, 7-33
master out slave in (MOSIx) pins, 7-5
master-slave interconnections, 7-3
multiplexing, 7-4, 14-17
open drain output enable (OPD) pin, 7-17
operation, master mode, 7-36
operation, slave mode, 7-39
operations, 7-33
packed data transfers, 7-18
polarity, clock, 7-14, 7-26
programming examples, 7-44
receive data (RXSPI) buffer, 7-7, 7-34
registers, A-14
send zero (SENDZ) bit, 7-20, 7-34
slave boot mode, 14-41
slave mode, 7-24, 7-35, 7-36
$\overline{\text{SPIDS}}$ pin, 7-35, 7-38

SPI *(continued)*
switching from receive to transmit mode, 7-41, 7-42
system, configuring and enabling bits, 7-37, A-14
TCB, 2-31
throughput, 7-7
transfer formats, 7-14
transmit data (TXSPI) buffer, 7-7
transmit underrun error (SPIUNF) bit, 7-29, 7-30, 7-31, A-19
TXFLSH (flush transmit buffer) bit, 7-42, A-17
unpacking data, 7-18
SPI bits
chained DMA enable (SPICHEN_A and SPICHEN_B), 7-37, A-19
chain loading status (SPICHS), A-20
clock phase (CPHASE), A-16
clock polarity (CLKPL), A-16
device select enable (DSxEN), 7-36
DMA interrupt enable (INTEN), 7-29
enable interrupt on error (INTERR), 7-29
enable (SPIEN), A-17
error status, 7-21
FIFO clear (FIFOFLSH), A-19
flush receive buffer (RXFLSH), 7-42, 7-43, A-17
flush transmit buffer (TXFLSH), A-17
get more data (GM), 7-20, 7-34, A-16
input slave select enable (ISSEN), 7-19
input slave select (ISSEN), A-16
internal loop back (ILPBK), A-17
low priority interrupt (SPILI), 7-29
master select (SPIMS), A-16
MISO disable (DMISO), A-16
mode fault error (MME), 7-19
most significant byte first (MSBF), A-16
open drain output select (OPD), A-17
SPI bits *(continued)*
packing enable (PACKEN), A-17
receive overflow error (SPIOVF), 7-29, 7-30, 7-31
seamless transfer (SMLS), A-17
send zero (SENDZ), A-15
SENDZ (send zeros) bit, 7-20, 7-34
sign extend (SGN), A-17
SPIPDN (SPI enable) bit, A-9
word length (WL), A-16
SPICHEN_A and SPICHEN_B (SPI DMA chaining enable) bits, 7-37, 7-39, A-33, A-37, A-41
SPICLK (SPI clock) pins, 7-14
SPICLK (SPI clock) signal, 7-5
SPICTL (SPI port control) registers, A-14
SPIDMAC (SPI DMA control) register, 7-23, A-18
$\overline{\text{SPIDS}}$ (SPI device select) pin, 7-14
$\overline{\text{SPIDS}}$ status, *See* ISSS bit
SPI interface signals, 7-10
SPILI (SPI low priority interrupt) bit, 7-29
SPI master mode operation, 7-33
SPIOVF (SPI receive overflow error) bit, 7-29, 7-30, 7-31
SPI registers
DMA configuration (SPIDMAC), 7-23, 7-37, 7-39, 7-41, 7-42, A-18
DMA control (SPIDMACx), 7-21
flag (SPIFLGx), A-22
interrupt latch (IRPTL), 7-29
interrupt latch/mask (LIRPTL), 7-29
interrupt (LIRPTL), 7-29
multiplexing, 14-17
receive buffer (RXSPI), 2-9
receive control (SPICTL, SPICTLB), A-14
RXSR (SPI receive shift), 7-5
SPIBAUD (baud rate) register, A-20

SPI registers *(continued)*
status (SPISTAT), 7-17, 7-19, 7-41, 7-44, A-20
status (SPISTAT, SPISTATB), A-20
transmit buffer (TXSPI), 7-20, 7-34, A-20
TXSR (SPI transmit shift), 7-5
SPISTAT, SPISTATB (SPI status) registers, A-20
SPIUNF (SPI transmit underrun error) bit, 7-29, 7-30, 7-31
SPORT bits
chained DMA enable (SCHEN), 6-50, 6-51, 6-57, A-33, A-37, A-41
channel A enable(SPEN_A), 6-37, 6-39, A-32, A-36
channel B enable (SPEN_B), 6-37, 6-39
channel error status (DERR_A), A-35
channel error status (ROVF_A or TUVF_A), 6-12, 6-15, A-39
clock, internal clock (ICLK), MSTR (I^2S mode only), A-32
clock, MSTR (I^2S mode only), A-37, A-40
clock rising edge select (CKRE), 6-25, 6-28
control bit comparison, 6-11
data buffer error status (DERR_x), 6-16
data independent transmit/receive frame sync (DIFS), A-33, A-37
data type (DTYPE), 6-11, 6-20, 6-22
DMA enable (SDEN), A-33, A-37, A-41
DXS_B (data buffer status), A-35, A-38, A-42
frame on rising frame sync (FRFS), 6-37, 6-39
frame sync delay (MFD), 6-45
FS both enable (FS_BOTH), A-34
SPORT bits *(continued)*
internal frame sync select (IFS), A-33, A-41
internal serial clock (ICLK), 6-25, 6-27
late frame sync (LAFS), A-33, A-37
number of channels (NCH), 6-45
operation mode (OPMODE), 6-11, 6-19, 6-28, 6-34, 6-37, 6-39, 6-44
receive underflow status (ROVF_A or TUVF_A), A-35, A-39
serial word length (SLEN), 6-20, 6-35, 6-37, 6-39, 6-44
SPORT modes, 6-34
(I^2S), 6-18
I^2S (Tx/Rx on left channel first), 6-19
I^2S (Tx/Rx on right channel first), 6-19
left-justified, 6-19, 6-36, C-5
loopback, 6-54
multichannel, 6-40
multichannel- A and B channels, 6-20
standard DSP, 6-19, 6-34, C-2
SPORT registers, 6-10
channel selection, 6-45
control, 6-11
control (SPCTLx), 6-9, 6-10, 6-11, 6-12
divisor (DIVx), 6-9
multichannel control (SPMCTLxy), 6-13, 6-19, 6-45, 6-54
receive buffer (RXSPx), 6-10, 6-13, 6-14, 6-15, 6-23, 6-38, 6-47, C-4
receive compand (MRxCCSx), A-46
receive select (MRxCSx), A-46
SPCTLx (serial port control), A-30
transmit buffer (TXSPx), 6-10, 6-13, 6-14, 6-16, 6-23, 6-38, 6-43, 6-47, C-4
transmit compand (MTxCSx, MTxCCSx), A-45

SPORTs
See also SPORT bits, modes, registers
128-channel TDM, 6-5
active low versus active high frame syncs, 6-30
buffer error status, 6-16
buffer hang disable (BHD) bit, 6-55
buffers, data, 6-13
chain insertion mode (DMA), 6-52
channel number (quantity) select (NCH bit), 6-45
clock, master, 6-25
clock divisor equation, 6-25
clock edge selection, 6-28
clock frequency equation, 6-25
clock (SCLKx) pins, 6-9
clock signal options, 6-25 to 6-28
companding and data type bit (DTYPE) 6-23
companding (compressing/expanding), 6-5
configuring frame sync signals, 6-9
configuring standard DSP serial mode, 6-34
control bit comparison, 6-11
data type, sign-extend, 6-22
data type, zero-fill, 6-22
divisor (DIVx) register, 6-25, 6-27, 6-30, 6-35, 6-36
DMA chaining, 6-51
DMA channels, 6-49
duplex, full, 6-9
endian format, 6-21
equation
frame sync frequency, 6-26
examples, normal vs. alternate framing, 6-32
features, 6-4
flag pins, 6-10
FLAGx pins, 6-10

SPORTs *(continued)*
framed and unframed data, 6-29
framed vs. unframed data example, 6-32
frame sync and serial word length, 6-26
frame sync delay, 6-45
full-duplex operation, 6-9
input/output (FLAGx) pins, 6-10
internal clock selection, 6-25
internal serial clock setting, 6-36
interrupts, 6-50, 6-53
I/O processor bus and, 6-21
latency in writes, 6-55
left-justified mode control bits, 6-37
loopback mode, 6-54
masking interrupts, 6-53
master mode enable, 6-39
multiplex data channels, 6-38, 6-49
operation modes, changing, 6-10
operation modes, listed, 6-18
operation modes, standard DSP serial, C-2
packing enable (PACK) bit, 6-11, 6-20, 6-21, 6-50, 6-53
pairing, 6-41
programming examples, 6-57
pulse code modulation (PCM), 6-38
receive buffers, 6-15
serial clock pins, 6-9
serial word length and frame sync, 6-26
setting frame sync rates, 6-36
signal sensitivity, 6-8
SPORTx_DA and SPORTx_DB channel data signal, 6-9
SPORTx_FS (serial port frame sync) pins, 6-9
TCB, 2-30
throughput, 6-2
transmit buffers, 6-13
transmit underflow status (TUVF_A) bit, A-35, A-39

Tx/Rx on FS falling edge, 6-19
Tx/Rx on FS rising edge, 6-19
using with SRU, 6-6
word length, 6-20
SPTRAN (serial port data direction control) bit, A-34, A-38
SRAM, 1-2, 4-2, 4-8, 4-22, 14-32, 14-48
SRAM memory address range, 4-8
SRC
block diagram, 12-5
clocking, 12-15, C-7
configuring modes, 12-12
control (SRCCTLx) register, 12-12, A-60
data paths, 12-9
data ports and, 12-12
de-emphasis (DEEMPHASIS) bits, 12-9
frame sync signal, 12-4, 12-9
interrupts, 12-19
mute (MUTE_OUT/MUTE_IN) signals, 12-9
mute (SRCMUTE) register, 12-9, A-64
muting, 12-9
ratio (SRCRAT) register, A-64
right justified mode, 12-12, 12-18
right-justified mode, 12-12, 12-13
right-justified mode, timing, 12-13
sample rate ratio, 12-9
sample rates, input, 12-9
time division multiplexing mode, 12-15, 12-18, C-8
SRC bits
auto mute (SRC0_AUTO_MUTE), A-60
bypass (SRC0_BYPASS), A-61
de-emphasis (SRC0_DEEMPHASIS), A-61
dither select (SRC0_DITHER), A-61
enable (SRC0_ENABLE), A-62
SRC *(continued)*
hard mute (SRC0_HARD_MUTE), A-60
matched phase select (SRC0_MPHASE), A-62
serial input format (SRC0_SMODEIN), A-61
serial output format (SRC0_SMODEOUT), A-61
soft mute (SRC0_SOFTMUTE), A-61
word lenght, output (SRC0_LENOUT), A-62
SRU
See also DAI
bidirectional pin buffer, 5-6
buffers, 5-6
clock routing control registers (Group A), A-80
connecting peripherals with, 5-13
connecting through, 5-16
connection to precision clock generator (PCG), 13-2
frame sync routing control registers (Group C), A-89
frame sync routing control (SRU_FSx) registers, A-89
group A (clock) signals, 5-20
inputs, 5-13
miscellaneous signal routing registers (Group E), A-98
outputs, 5-13
pin buffer enable registers (Group F), A-102
pin signal assignment registers (Group D), A-93
registers, A-80 to A-105
register use of, 5-16
serial data registers (Group B), A-85
serial ports and, 6-6
signal groups, 5-14

SRU *(continued)*
signal groups, defined, 5-13
signal sources, clock, A-80
signal sources, frame sync, A-89
signal sources, miscellaneous, A-98
signal sources, pin signal, A-93
S/PDIF configuration, 11-11
SPORT signal connections, 6-6
SRU registers
clock (SRU_CLKx), A-80
frame sync (SRU_FSx), A-89
miscellaneous (SRU_EXT_MISCx), A-98
overview, A-80
pin assignment (SRU_PINx) registers (group D), A-93
pin enable (SRU_PINENx) registers, A-102
pin signal (SRU_PINx), A-93
SRU_DATx (SRU data) registers, A-85
SRU_EXT_MISCx (SRU external miscellaneous) registers, A-98
SRU_FSx (SRU frame sync routing control) registers, A-89
SRU_PINENx (SRU pin buffer enable) registers, A-102
SRU_PINGx_STAT (ping-pong DMA status) register, A-55, A-56, A-80
SRU_PINx (pin signal assignment) registers, A-93
stalls, processor, 2-32
standard DSP serial mode, C-2
starting an interrupt driven transfer, 8-27, 8-28
STROBEA (one shot frame sync A) bit, 13-14, A-67
STROBEB (one shot frame sync B) bit, 13-14, A-68
strobe period, 13-15
strobe pulse, 13-14
supervisory circuits, 14-26
support, technical or customer, xxxvi
switching from receive to transmit DMA, 7-42
switching from transmit to receive DMA, 7-41
synchronization with the external clock, 13-9
synchronizing frame sync output, 13-9
SYSCTL (system control) register, A-4
system, 14-2, 14-28
system design
baud rate, init value, 14-39
block diagram, 1-3
bypass capacitors, 14-29
CLKIN pin, 14-5
clock distribution, 14-28
clocking, 14-4
clock input, 14-5
conditioning input signals, 14-2
crosstalk, 14-29
decoupling capacitors, 14-29
designing for high frequency operation, 14-28
ECPP register, 14-38
EIPPx registers, 14-38
EMPP register, 14-38
generators, reset, 14-27
ground plane, 14-29
hold time, inputs, 14-4
input setup and hold time, 14-4
input signal conditioning, 14-2
JTAG interface pins, 14-16
latchup, 14-2
latency, input synchronization, 14-18
maximum clock frequency, 14-6
parallel port, DMA address (IMPP) register, 14-38
parallel port, DMA external address (EIPPx) registers, 14-38

system design *(continued)*
parallel port, DMA external word count (ECPP) register, 14-38
parallel port, DMA internal word count (ICPP) register, 14-38
parallel port DMA external address (EMPP) register, 14-38
pin descriptions, 14-16
plane, ground, 14-29
PLL start-up, 14-11
power options, 14-9, 14-13
power supply, monitor and reset generator, 14-27
power-up, 14-11
recommendations and suggestions, 14-29
RESET pin, 14-2
VCO encodings, 14-9

T

TCB
chain loading, 2-21
defined, 2-20
memory storage, 2-21
parallel port, 2-30
sizes, 2-21
SPI, 2-31
SPORT, 2-30
structure, 2-22
TCK (test clock) pin, 14-16
TDI (test data input) pin, 14-16
TDO (test data output) pin, 14-16
technical or customer support, xxxvi
technical publications on the web, xl
technical support, xxxvi
test mode
DAI use in, 5-10, 5-29
data buffer use in, 2-10
loopback, SPI, 7-33, A-17
SPI, 7-32
test mode *(continued)*
SPORT, 6-24, 6-54, 9-21, A-44
system, 14-2, 14-28
throughput
IDP, 8-6
memory-to-memory port, 3-3
parallel port, 4-19, 4-21, A-12
SPI, 7-7
SPORTs, 6-2
time division multiplexed (TDM) mode, 6-40, 12-9, 12-15, C-7
time division multiplexed (TDM) mode (S/PDIF), 11-11
timer *See* peripheral timers, core timer
timing
asynchronous, 14-3
IDP I^2S, 8-8
IDP left-justified, 8-8
input clock, 14-6
input clock divider (INDIV bit), 14-6
kernel boot, 14-47
parallel port, 4-4, 4-20
PWM, 10-13, 10-15
S/PDIF, 11-4
SPI clock, 7-31, 7-32
SPI slave, 7-31
SPI transfer protocol, 7-15
SPORT bits, 6-34
SPORT framed vs. unframed data, 6-32
SPORT frame sync, 6-36
SPORT I2S, 6-39
SPORT left-justified, 6-37
SPORT normal vs. alternate framing, 6-32
SRC, 12-13
TIMOD (transfer initiation mode) bit, 7-29, 7-34
TMSTAT (peripheral timer global status and control) register, 9-7
TMS (test mode select) pin, 14-16

TMxCNT (peripheral timer word count) registers, 9-5, 9-6
TMxCTL (peripheral timer control) registers, 9-5, A-56
TMxPRD (peripheral timer period) registers, 9-6
TMxSTAT (peripheral timer status) register, A-57
TMxW (peripheral timer width) registers, 9-6
TMxW (peripheral timer word pulse width) registers, 9-5
T_PRDHx (timer period) registers, 9-5, 9-6
transfer control block, *See* DMA TCB
transfer initiation and interrupt (TIMOD) mode, 7-29
transmission error (TUNF) bit, 7-20
transmit and receive channel order (FRFS), 6-39
transmit and receive SPORT data buffers (TXSPxA/B, RXSPxA/B), 6-13
transmit collision error (TXCOL) bit, 7-20
transmit data (TXSPI) buffer, 7-7
transmit FIFO status, 4-18, 4-24
transmit shift (TXSR) register, 7-5
$\overline{\text{TRST}}$ (test reset) pin, 14-16
TUNF (transmission error) bit, 7-20
TUVF_A (channel error status) bit, A-35, A-39
two channel mode (S/PDIF), 11-8
TXCOL (transmit collision error) bit, 7-20
TXFLSH (flush transmit buffer) bit, 7-42, A-17
TXS_A (data buffer channel B status) bit, A-35, A-39, A-42
TXSPI, TXSPIB (SPI transmit buffer) registers, A-20
TXSPI (SPI transmit buffer) register, 2-9, 7-20, 7-22, 7-34
TXSPx (serial port transmit buffer) registers, 2-9
TXSR (SPI transmit shift) register, 7-5

U

unpacking sequence for 32-bit data, 4-7
update modes, (PWM), 10-20

V

VCO, 14-4, 14-5, 14-6
  bypass clock, 14-10
  clock, 14-8, 14-9
  examples, clock management, 14-14
  output clock, 14-9
voltage controlled oscillator, *See* VCO

W

warning
  clocks and system clocking, 14-4, A-7
  I/O processor, 2-20, 2-32
  parallel port, 4-12, 4-13
  SPI, 7-23, 7-41
  SPORTs, 6-5, 6-14, 6-17, 6-43
watchdog function, timer, 9-20
WDTH_CAP (width capture) mode, 9-2, 9-14
word length, 6-20
word length (SLEN) bits, 6-37, 6-39
word packing enable (packing 16-bit to 32-bit words), A-32, A-37, A-40
write cycle, parallel port, 4-5
$\overline{\text{WR}}$ (write strobe) pin, 4-9